Universidad Autónoma de Madrid

Trabajo de Fin de Master

Estudio del procesamiento de información en peces eléctricos mediante técnicas lazo cerrado de estimulación en tiempo real

Autor:
Ángel Lareo Fernández

Tutor:
Francisco de Borja Rodriguez Ortiz

Máster Universitario en Investigación e Innovación en TIC

Grupo de Neurocomputación Biológica
Departamento de Ingeniería Informática

26 de septiembre de 2014

Resumen

Estudio del procesamiento de información en peces eléctricos mediante técnicas lazo cerrado de estimulación en tiempo real

Los peces de la especie Gnathonemus Petersii hacen uso de la electrorrecepción activa (generación y detección de campos eléctricos) para la comunicación con otros individuos y la localización de objetos. Dichos procesos exigen un procesamiento de la información (codificación, decodificación...). El objetivo de este proyecto es comprender qué procesos básicos guían la actividad eléctrica del pez a la hora de comunicarse y orientarse.

Para ello, se ha adaptado una plataforma hardware-software dedicada en tiempo real para la adquisición de la señal emitida por el sistema biológico y la estimulación artificial del mismo mediante técnicas en ciclo cerrado operando en tiempo real. Esto hace posible una interacción optima y controlada entre los dispositivos artificiales y los sistemas de procesamiento de información naturales del pez. Este modelo en ciclo cerrado resulta fundamental para caracterizar y extraer información del sistema en vivo. Además, en dicha plataforma se han implementado una serie de módulos y mejoras que permiten un mayor control de la actividad del pez y el uso de nuevos protocolos de estimulación en ciclo cerrado.

Por otro lado, el uso de técnicas de teoría de la información ha permitido tanto caracterizar la actividad eléctrica del pez como estudiar cambios en el procesamiento de información. Dado un modelo de codificación de la información sobre la señal eléctrica del pez, el cálculo de la entropía sobre la serie de códigos resultante permite estudiar la capacidad de transmisión de información de dicha señal. Se ha realizado un estudio entrópico de la evolución de la actividad eléctrica del pez sin estimulación en periodos largos de tiempo que muestra variaciones en la entropía de la señal incluso en ausencia de estimulación artificial. Este estudio nos permite además seleccionar los parámetros para la digitalización que guiarán el establecimiento del ciclo cerrado en la estimulación dependiente de actividad dirigida por códigos.

También se ha realizado un análisis (basado en el uso de una codificación digital binaria) sobre la influencia de una memoria a corto plazo. Mediante la implementación de un módulo de digitalización y binarización on-line de la señal adquirida, ha sido posible caracterizar la información en forma de códigos binarios y establecer en función de ellos un ciclo cerrado de estimulación. Los resultados de dicho análisis muestran variaciones en el procesamiento de información cuando la estimulación depende de la actividad eléctrica del pez en un rango de tiempo, lo que da indicios prometedores de la influencia de la memoria en el procesamiento de información.

Finalmente, el uso de metodología estadística aplicada a la teoría matemática de la comunicación nos permite un análisis no lineal de los resultados, necesario para extraer información sobre los cambios en el procesamiento de información en función de la estimulación presentada.

Abstract

Information processing in weakly electric fishes through closed-loop stimulation in real time

Active electro-reception (the generation and detection of electric fields) is used by Gnathonemus Petersii fish in order to communicate with other individuals and to locate objects. Such processes require an information processing (code, decode...).

The aim of this work is to understand which processes guide the fish's electric activity so it would be able to communicate and orientate. During our investigation we have used a hardware-software platform to collect the signal emitted by the biological system and also to stimulate it by closed-loop techniques that operate in real time. This method allows a positive and controlled interaction between the artificial mechanisms and the fish's natural systems of information processing. Such a closed-loop model is essential in the in vivo characterization and extraction of information from the fish. Additionally we have manage to implement a certain amount of modules and improvements on the platform which enable a better control of the fish's activity and the use of new closed-loop stimulation protocols.

On the other hand, the characterization of the fish's electric activity and the study of the changes done during the information processing as well have been made possible by using techniques from the information theory. In that order, given a coding model of information over the fish's electric signal and having the entropy of the groups of codes calculated, the result evaluates the signal's capacity of data transmission. We have also made an entropic study of the development of the fish's activity with no stimulation during long time periods that shows entropy variations even in absence of stimulation. This study has enabled us to select the parameters for the digitalization which will lead the establishment of the closed-loop along the stimulation that depends on the activity run by codes.

Furthermore, we have done an analysis about the influence of a short-time memory based on a binary digital coding. Through the implementation of digitalization and binarization on-line module of the collected signal, it has been possible to characterize the information into binary codes by mean of which we have established closed-loop stimulation. These results show variations in the information processing when the stimulation takes short-time range electrical activity into account, which provide promising indications about the influence of the memory.

In the end, using statistics applied into the mathematical theory of communication has allows us to make a non-lineal analysis of our study's results. Such type of research is

essential for the extraction of information about the changes along the data processing which depends on the stimulation appeared.

Índice general

Índice de figuras

Índice de cuadros

Siglas

DAQ	**D**ata **A**c**Q**uisition
GNB	**G**rupo de **N**eurocomputación **B**iológica
OE	**Ó**rgano **E**léctrico
IPI	**I**nter-**P**ulse **I**nterval
PCB	**P**rinted **C**ircuit **B**oard
RTAI	**R**eal-**T**ime **A**pplication **I**nterface
SO	**S**istema **O**perativo
STR	**S**istema en **T**iempo **R**eal

Agradecer a mi tutor, Francisco de Borja, por la confianza, los consejos, la motivación y todo el esfuerzo invertido. A Pablo por el apoyo y el interés. A Alejandro, por cuidar la dieta de los peces. A Fernando, por guiarme en mis primeros días en el GNB y conseguir que siempre aprendiese algo. Por supuesto al resto de personas del GNB: Irene, Aarón, Carlos, Ana, Miguel... Por sus conocimientos y su ayuda, pero sobre todo por los buenos momentos.

Mención especial a Víctor porque su ilusión (y sus placas de circuito impreso) sí que nos han mantenido a flote.

A mis padres, por el apoyo material, y a mis hermanas por el emocional. A Marta y Yeray por llenarme de vida. También a mis otros hermanos: A J, a Lau y a Nacho, no sé dónde estaría sin vosotros, pero seguro que estaría peor.

Un agradecimiento enorme para Rebeca por su pasión biológica y su esfuerzo electrorreceptivo.

A Kairós, porque le debo todo lo que falta en este trabajo.

Y, en fin, a todos aquellos que se han preocupado, desde el primero al último, que son unos cuantos.

Capítulo 1

Introducción

Los procesos de comunicación en organismos vivos pueden ser muy diversos, como resultado de los procesos evolutivos y debido a los mecanismos de adaptación de las especies al medio en el que habitan. Si entendemos por comunicación la transmisión de algún tipo de información desde un ente (o nodo) a otro, esta puede tener lugar mediante técnicas muy diferenciadas. Por ejemplo, haciendo uso de ondas en medios como el aire (caso de la comunicación sonora) o bien mediante la dispersión de compuestos químicos (caso de la comunicación mediante feromonas u odorantes). De manera general, el estudio de los procesos de comunicación biológicos puede proveernos de importantes lecciones para el desarrollo de nuevos métodos de comunicación artificial

En este trabajo nos centraremos en el estudio de la comunicación mediante el uso de campos eléctricos en el medio acuático. Dicho sentido eléctrico, conocido como electrorrecepción[2], permite la transmisión de información de organismos, en un proceso conocido como electrocomunicación, y la obtención de información del medio y los objetos presentes en el, lo que se conoce como electrolocalización. Este sentido eléctrico permite a determinados organismos orientarse y comunicarse en entornos de visibilidad reducida y su comprensión tiene una aplicabilidad práctica muy diversa. Como ejemplo se puede nombrar la utilización de peces eléctricos para detectar la calidad del agua [3], de manera que se monitoriza la actividad eléctrica del pez habitando en determinadas aguas y en función de esta se puede analizar la presencia de sustancias contaminantes. Otro ejemplo podría ser el desarrollo de robots subacuáticos bioinspirados en la electrolocalización para resolver los problemas de accesibilidad y visión en entornos de aguas turbias o con poca luz [4–7].

Para que el proceso comunicativo pueda tener lugar, este exige un procesamiento previo de la información a transmitir (codificación) y un procesamiento posterior de la información recibida (decodificación). Por tanto, el estudio de los mecanismos biológicos de

comunicación parte del estudio de cómo la información es procesada en los organismos. En el caso de este trabajo estudiamos el procesamiento de la información eléctrica en peces eléctricos de descarga débil electroactivos. Más en concreto, de la especie Gnathonemus Petersii, propia del África Occidental [8]. Si bien este estudio puede beneficiar el de otros sistemas biológicos, ya que la metodología desarrollada puede aplicarse al estudio del procesamiento de información en una variedad de sistemas, tanto de otras especies que hacen uso de la electrorrecepción [9] como de otros mecanismos comunicativos.

Este organismo, en el proceso de electrorrecepción hace uso de un órgano eléctrico (OE) situado en su aleta caudal, en la parte posterior de su cuerpo. Este órgano le permite polarizar su cuerpo, lo que modifica el campo eléctrico en su entorno de manera que puede ser percibido como descargas (cambios rápidos en el potencial eléctrico en torno al pez). Dichas descargas generan una señal pulsar como la que puede observarse en la figura 1.2, que le sirven al pez para comunicarse con otros especímenes y para orientarse en su entorno. Más información sobre la electrorrecepción se detalla en la sección 1.1.

Los pulsos presentes en la señal emitida por el pez se encuentran totalmente estereotipados en cada especimen, por lo que la información que estos transmiten se codifica no en la forma del pulso sino en otras características de la señal, como puede ser el tiempo entre pulsos. Lo que se conoce como Inter-pulse Interval o IPI [10–12]. La idea de que la distribución de IPIs a lo largo de la señal emitida por el pez aporta información sobre el estado del pez y es por tanto un modo de codificar la información en la señal eléctrica ya ha sido utilizada en otras ocasiones [12]. Es el objetivo de este trabajo estudiar el modo en que la información se procesa en la señal haciendo uso, entre otros, de este concepto y de protocolos de teoría de la información, tal como se detalla en la sección 1.2.

Por último, resulta esencial para el estudio descrito el uso de mecanismos ciclo cerrado para la estimulación del sistema biológico. Los mecanismos tradicionales son de tipo monodireccional y consisten en la reproducción de estímulos pregrabados de la señal electrorreceptiva del pez en diversos estados (reposo o en ataque) o incluso algún tipo de señal completamente artificial basada en una secuencia de pulsos. Esta estimulación tradicional permite la caracterización de ciertas respuestas al pez, pero no permite una variación online de los estímulos en función de la respuesta observada. El modelo en ciclo cerrado, al contar con la realimentación de la respuesta que obtiene del sistema biológico, permite que se revelen dinámicas de procesamiento de información propias de un proceso bidireccional de recepción y emisión de estímulos como el que constituyen de manera natural buena parte de los procesos comunicativos. Ese tipo de dinámicas permanecerían ocultas bajo un modelo de estimulación tradicional. La estimulación en ciclo cerrado ha sido ampliamente utilizada para el estudio de sistemas biológicos [1]. A

nivel neurofisiológico, por ejemplo, bajo el concepto de Dynamic Clamp [13]. La motivación del uso de ciclos cerrados, así como su utilización en investigaciones diversas y su aplicación concreta al caso que nos ocupa se encuentran desarrolladas en la sección 1.3.

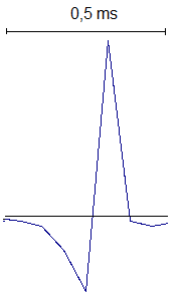

FIGURA 1.1: Un pulso simple

1.1. Electrorrecepción

Los sistemas sensoriales de los seres vivos son muy selectivos. Los procesos evolutivos permiten que los animales, con capacidades sensoriales variables, se adapten a un medio concreto para su supervivencia. Entre las diferentes adaptaciones sensoriales encontramos la capacidad de algunos seres vivos para detectar, generar y utilizar campos eléctricos. A esta capacidad se le denomina electrorrecepción [13] y aparece por primera vez hace más de 500 millones de años en muchas ramas de vertebrados ancestrales. Esta ha sido observada casi por exclusividad en animales de ambientes acuáticos o relacionados íntimamente con el agua (peces y anfibios generalmente), ya que las propiedades conductoras del agua frente al aire facilitan la transmisión eléctrica y por lo tanto, la mayoría de células especializadas en la electrorrecepción son acuáticas. Esta capacidad ha aparecido de manera independiente en diferentes ramas evolutivas, por lo que se deduce que es una adaptación beneficiosa para la supervivencia de especies que habitan en ambientes acuáticos [14, 15]. Algunas especies tienen, también, la capacidad de generar campos eléctricos, es decir capacidad electrogénica. Esta capacidad es posible gracias a la presencia de un conjunto de células eléctricas derivadas de células musculares (o neuronales en el caso de la familia de los Apteronotidinae) que forman el órgano eléctrico, encargado de polarizar el cuerpo del pez [2]. La electrogénesis tiene una historia evolutiva más reciente, si nos atenemos a la historia evolutiva de los peces cartilaginosos. En los peces óseos, la electrogénesis y la electrorrecepción se desarrollan múltiples veces y de forma convergente durante la evolución [14], de manera que peces con capacidad eléctrica alejados taxonómicamente entre sí, poseen mecanismo de producción y recepción de descargas similares [15].

Las células electrorreceptoras son las encargadas de la percepción y entrada en el sistema nervioso de las señales eléctricas, y se encuentran distribuidas por toda la epidermis del animal [2, 5, 16, 17] Se pueden clasificar en dos grupos principales según sus características fisiológicas y morfológicas: en ampolla y tuberosos [18]. Los receptores en ampolla responden de manera más efectiva a las sinapsis eléctricas de baja intensidad, en corriente continua. En la mayoría de las especies con capacidad electrorreceptora los sensores eléctricos o electrorreceptores son de este tipo. En cuanto a los tuberosos, estan especializados en señales de frecuencia más altas, especialmente en las frecuencias dominantes producidas por el órgano eléctrico (OE), y se ha encontrado una única excepción de peces con este tipo capaces de electrogénesis.[14]

En lo referido al tipo de señal eléctrica producida por el OE, los peces eléctricos se pueden dividir en dos grupos: onduladores y pulsares. Los onduladores producen descargas continuas modulando su frecuencia y amplitud. Por otro lado los pulsadores producen pulsos cortos estereotipados (generalmente del orden de ms de duración) con intervalo entre pulsos, IPI (inter pulse interval), entre 5-600ms que pueden ser modulados de acuerdo a estímulos del ambiente [19] . La duración de los pulsos es bastante similar entre los animales de la misma especie pero suele variar considerablemente entre individuos de distintas especies [2]

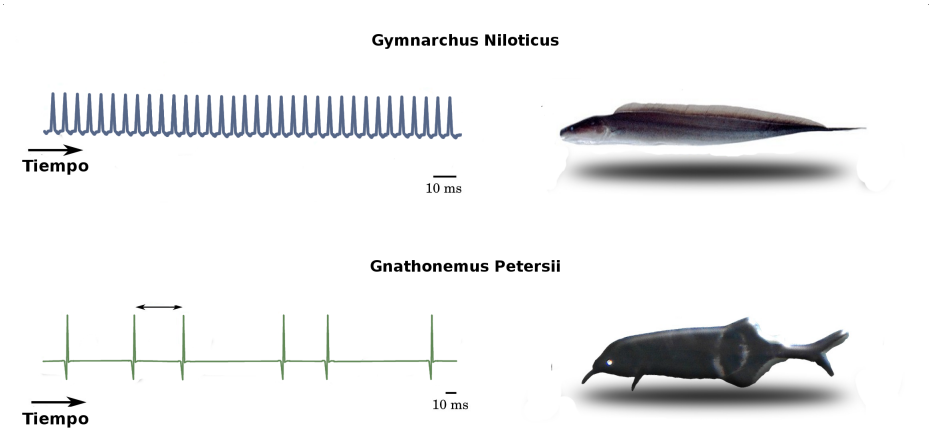

FIGURA 1.2: Tipos de señales emitidas por peces eléctricos: Sinusoidal y pulsar.

En el proceso de electrorrecepción el OE produce un campo eléctrico alrededor del pez similar a un dipolo, de forma que los objetos que se encuentran en dicho campo eléctrico alteran la corriente inducida a los electrorreceptores, provocando que el pez pueda construir una imagen eléctrica.[20] A esta forma de electrorrecepción se le denomina electrorrecepción activa. De esta forma los electrorreceptores detectan los cambios y distorsiones en el campo eléctrico generado alrededor del cuerpo del pez facilitando su movimiento en condiciones de penumbra, ambientes nocturnos o aguas turbias [21–23].

El proceso de análisis de las alteraciones de los campos eléctricos autoproducidos del pez monitorizado se denomina electrolocalización. [5, 20, 24–27] En este proceso, los objetos con menor impedancia que el agua atraen las corrientes eléctricas, ya que estas fluyen mejor a través de objetos de impedancia baja, llevando una mayor densidad de corriente que penetra en la piel del pez. El efecto opuesto ocurre cuando los objetos tienen una impedancia mayor que la del agua, es decir, repelen las corrientes eléctricas y entra una densidad de corriente menor en la piel del animal [7].

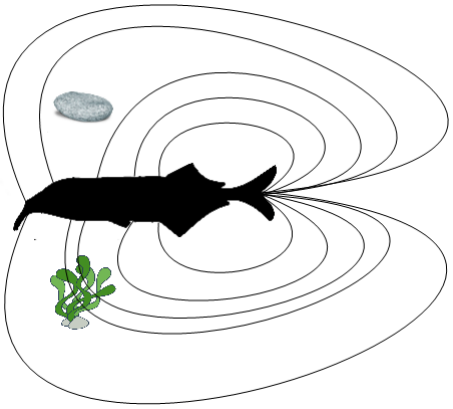

FIGURA 1.3: Alteracion del campo electrico por la variacion de impedancia de los objetos respecto al agua

Además de para la electrolocalización estos peces también usan la electrorrecepción activa para la electrocomunicación, es decir, para comunicarse socialmente, identificar el sexo o tamaño de otros individuos y resolver disputas territoriales. Este mecanismo consiste en la emisión y recepción de señales eléctricas entre dos o más peces, que forman un canal eléctrico de forma que cada pez tiene su campo eléctrico distorsionado y una corriente transcutanea producidas por las descargas de los otros peces que intervienen en el proceso.[17, 19, 28, 29].Los canales eléctricos formados en la electrocomunicación son específicos a nivel de especie ya que se crean "canales privados" de comunicación basados en que el individuo de una determinada especie emite impulsos muy similares a los que utiliza para la electrolocalización (con frecuencias de emisión más altas) de forma que son recibidos por los electrorreceptores específicos de otro individuo que los interpretan en función del IPI y la longitud del pulso. De esta forma los peces pueden determinar aspectos como el sexo o el dominio territorial e iniciar procesos de cortejo o ataque [30–34].

Tanto la electrolocalización como la electrocomunicación son mecanismos muy complejos ya que precisan de un procesamiento de una gran cantidad de información sensorial por parte del sistema nervioso de los peces eléctricos [35]. Los peces eléctricos se pueden dividir en dos grandes grupos dependiendo de las características de sus descargas: los fuertemente eléctricos, como las anguilas eléctricas, y los débilmente eléctricos, como

el pez elefante. Los peces fuertemente eléctricos como de él orden Siluriformes (bagres eléctricos) o Torpediniformes (rayas eléctricas) producen electricidad mediante un OE y son capaces de generar descargas de entre 220-500 voltios que utilizan como mecanismo de defensa y depredación. [2, 31] Por otro lado, los peces débilmente eléctricos, como el orden Gymnotiformes (con excepción del genero Electrophorus (anguilas eléctricas) que son peces fuertemente eléctricos) o Osteoglossiformes producen descargas eléctricas de baja intensidad de unos 5 voltios aproximadamente que utilizan para los procesos electrocomunicación y electrolocalización (explicados anteriormente). Este tipo de animales son adecuados para el estudio de la electrorrecepción ya que por su naturaleza es fácil adquirir su señal e implementar mecanismos de estímulo bidireccional. Por ultimo encontramos un tercer grupo, más extenso, incapaces de generar campos eléctricos (electrorrecepción pasiva) ya que carecen de órgano eléctrico. Estos utilizan su capacidad electrorreceptora para actividades como la depredación. Esto es posible gracias a que la actividad nerviosa y muscular de la presa genera cambios eléctricos en el medio que pueden detectados por el depredador. Son destacables en este ámbito los tiburones, que poseen células receptoras (ampolla de Lorenzini) capaces de detectar diferencias de hasta 5nv en bajas frecuencias [2, 36].

Dentro de los peces débilmente eléctricos encontramos el orden de los Osteoglossiformes, que posee dos familias: Gymnarchidae, con tan solo un ejemplar, gymnarchus niloticus (pez navaja africano, se ha utilizado como ejemplo de pez eléctrico de señal sinusoidal en la figura 1.2), y Mormyridae, una de las mayores familias de peces eléctricos con unas 200 especies. Los Osteoglossiformes poseen un OE capaz de producir señales eléctricas (capacidad electrogénica) y de recibir esas señales a través de electrorreceptores. [2, 5, 16, 17]

En la familia Mormyridae se encuentra incluido el género Gnathonemus que contiene a la especie Gnathonemus petersii especie elegida para estudio en este trabajo. Esta especie de Gnathonemus podemos encontrarla desde Nigeria, a través de la cuenca del Congo llegando al Norte de Angola y Zambia. [8]

El Gnathonemus petersii o pez elefante sobrevive en medios de reducida visibilidad gracias a su capacidad electrolocalizadora que utiliza para la búsqueda de alimento, el reconocimiento del medio y la navegación. Tambien se comunica con otros miembros de su especie gracias a la electrocomunicación, mecanismo que utiliza para intercambiar protocolos de cortejos o defensas territoriales [30–34].

El OE capaz de producir los impulsos eléctricos está formado por miles de células concentradas que se sitúan sobre la piel en la zona del pedúnculo caudal del pez y tiene un tamaño medio entre 1-2 cm. [34] Posee además electrorreceptores distribuidos por todo el cuerpo que le permiten detectar los impulsos generados por otros peces, o sus

FIGURA 1.4: Localizacion geografica del pez elefante

propios impulsos con la finalidad de comunicarse o detectar objetos u otros animales (electrocomunicación y electrolocalización). El OE es vital para la especie ya que es necesario para la orientación y búsqueda de alimento y también para los propósitos sociales tales como la localización de animales de su misma especie, su posición jerárquica y la búsqueda de individuos para la reproducción, punto fundamental para la supervivencia de la especie.[32]

Gnatonemus petersii es un pez pulsador, es decir, su descarga eléctrica es de tipo pulsar. La duración de los pulsos generados por el pez elefante son del orden de las decimas de milisegundo (0,8ms aproximadamente) y el IPI es de 10 a 600ms aproximadamente.

Los objetos cercanos al pez con una resistencia o conductividad diferente a la del agua alteran el campo eléctrico que percibe el pez. Las alteraciones del medio son detectadas por los electrorreceptores que se encuentran en la epidermis del pez. El pez elefante, al detectar estos cambios, interpreta y procesa la información generando la imagen eléctrica. [21, 36, 37] Al crearse la imagen eléctrica el pez reacciona ante el estímulo. Este proceso se repite durante toda la vida del pez que emite pulsos a una velocidad media de 150 pulsos por minuto, que disminuyen en casos de tranquilidad (como en territorio conocido o aislamiento social) y aumentan en situaciones de estrés.

La precisión de los receptores eléctricos de los peces elefantes es tan alta que permite estimar el valor de resistividad o capacidad de un objeto a su alcance dependiendo de las alteraciones del campo eléctrico que producen los objetos de su alrededor. La capacidad de discriminación eléctrica del pez está directamente relacionada con el tamaño del pez, el tamaño del objeto que produce la alteración y su conductividad con respecto al medio en el que se encuentra [6].

Estudios realizados de estímulos grabados asociados a procesos de ataque, peces en estado de reposo o secuencias aleatorias han comprobado que se producen distintas respuestas en estos peces en función del estímulo. La precisión temporal de los pulsos generados en respuesta es alta, y ya que está relacionada directamente con los procesos de comunicación, supone un punto a tener en cuenta de cara al diseño de experimentos que se utilicen para su estudio [12, 38, 39].

Los estudios realizados hasta ahora consisten en estimular siguiendo los protocolos básicos de unidireccionalidad, donde la reacción del pez no altera el estímulo emitido. Para simplificar los experimentos se utilizan, generalmente, estímulos artificiales como pulsos con formato de onda cuadrada o sinusoidal repetidos periódicamente. Estos procedimientos presentan grandes limitaciones ya que se ha observado una gran variabilidad en la distribución de IPIs entre análisis. También están siendo utilizadas en experimentos de electrofisiología técnicas de estímulo bidireccional dependientes del sistema desde la década de los 40 hasta la actualidad [40, 41].

La investigación básica sobre el procesamiento de la información en peces eléctricos resulta útil para la utilización de los peces como detectores de calidad de agua ya que los patrones de emisión se ven alterados con los contaminantes [3].

Por otro lado se está promoviendo el desarrollo de robots submarinos bioinspirados que se benefician del estudio de los sistemas naturales de electrolocalización, tal como el que utilizan estos peces, para implementar algoritmos de navegación a estima, búsqueda de animales o de lugares concretos. Este método de búsqueda es útil ya que puede resolver los problemas de accesibilidad y visión en entornos donde otros métodos de búsqueda fallan como por ejemplo en ambientes de aguas turbias o con poca luz [4–7].

1.2. Procesamiento de información

Como ya se ha comentado, los peces eléctricos de la especie *Gnathonemus Petersii* hacen uso de las EODs para percibir su ambiente, para buscar comida, identificar otras especies y para comunicarse. Contamos con un evento diferenciado en la señal eléctrica como son los pulsos emitidos por el pez, pero debido a que dicho evento se encuentra completamente estereotipado podemos asegurar que la información no se codifica en la forma, la amplitud o el tipo del pulso, y hemos de inferir que la información se encuentra codificada en la señal de algún otro modo. Por ejemplo, en base a la frecuencia de emisión o los tiempos entre pulsos (Inter-pulse intervals o IPIs, Figura 1.5). Estudios recientes han hecho uso de la distribución de IPIs para estudiar el procesamiento de

información en estos peces y cómo esta se ve modificada en función de la estimulación que se le presenta al sistema [10–12].

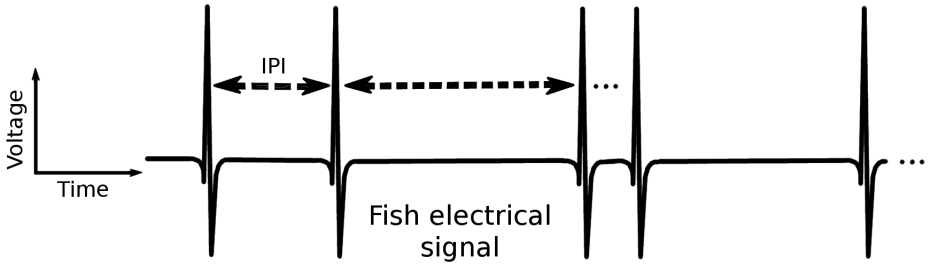

FIGURA 1.5: Inter-Pulse Interval.

Para el estudio sobre procesamiento de información haremos uso de técnicas de análisis estadísticas, como histogramas para mostrar distribuciones de probabilidad de IPIs y gráficos cuantil-cuantil que permiten comparar dos distribuciones y que podemos utilizarlos para analizar el comportamiento eléctrico del pez bajo condiciones diferentes de estimulación en un mismo experimento. Más información respecto al uso de estas y otras técnicas estadísticas se encuentra en la sección 2.2 sobre la metodología utilizada en este proyecto.

Respecto a la teoría de la información o teoría matemática de la comunicación, esta estudia la transmisión y el procesamiento de la información en sistemas de comunicación. Tiene su origen a finales de los años 40 [42] y encuentra aplicación en campos tan diversos como la criptografía [43] o la neurociencia [44].

Esta teoría nos resulta interesante para el estudio del procesamiento de la información ya que se ocupa de la medición de la información teniendo en cuenta la representación de la misma, de tal modo que permite estudiar la capacidad de diversos sistemas de comunicación para transmitir información en función de la codificación utilizada. Para la teoría de la información, la información (y la capacidad de transmisión de la misma) se caracteriza como una magnitud física que se puede calcular. Dado que esta capacidad es dependiente del modelo de codificación (representación) utilizado, el estudio con teoría de la información de señales destinadas a la comunicación nos aporta ciertas claves para inferir cómo la información es procesada en sistemas de los que desconocemos el modo en que tiene lugar dicho proceso, como es el caso que nos ocupa de la electrocomunicación en peces eléctricos.

Según esta teoría, un suceso no aporta información de manera aislada si no en base a la probabilidad P(x) de ese suceso de ocurrir en una serie, en concreto, la información de

un evento I(E) se define como

$$I(E) = \log \frac{1}{P(E)} \tag{1.1}$$

donde la base del logaritmo equivale a elegir una determinada unidad de información. Suele utilizarse $\log_2$ para expresar la información en *bits*. Un bit se define como la cantidad de información obtenida al especificar una de dos alternativas igualmente probables. Si bien si se desease expresar la magnitud equivalente de información en otra unidad bastaría con realizar un cambio de base logarítmica. Seleccionar los sucesos que vamos a tener en cuenta como portadores de información es equivalente a seleccionar un modelo de codificación de la información sobre la señal. En general una vez seleccionado un modelo de codificación, obtenemos la secuencia de símbolos resultante de aplicarlo a una señal adquirida.

Una vez seleccionada la codificación, podemos determinar la capacidad de transmisión de información mediante el cálculo de la entropía atendiendo a la frecuencia de aparición (o probabilidad) de cada símbolo en dicha secuencia de símbolos. En la teoría de la información, la entropía mide la cantidad de información promedio de los símbolos usados. Esta capacidad establece la información máxima que puede portar una secuencia de símbolos, lo que no supone que dicha secuencia de símbolos porte efectivamente dicha cantidad de información, es solo una medida de la capacidad máxima y como tal puede utilizarse como criterio de selección de una codificación determinada.

La fórmula para el cálculo de la entropía es la siguiente [45]:

$$H(X) = -\sum P(x) \log_2 P(x). \tag{1.2}$$

Mediante estas técnicas es posible realizar un análisis de la señal emitida por los peces eléctricos de descarga débil con el fin de determinar la capacidad de dicha señal para la transmisión de información. Esto nos permite analizar diferentes modelos de codificación de la información en la señal eléctrica de estos peces.

Otro de los objetivos de este proyecto es el estudio de la influencia de la memoria a corto plazo en el procesamiento de información. El uso de técnicas de teoría de la información para la realización de este objetivo ha sido muy importante. En la sección 1.2.2 se detallan las bases teóricas que guían la metodología escogida, cuyo funcionamiento se especifica en la sección 3.4.

Por último, es de notar que la teoría de la información también nos aporta técnicas para estudiar la dependencia entre señales. Esto puede realizarse mediante el cálculo de la información mutua, lo que nos aporta datos sobre la correlación entre dos eventos. Dicha técnica ya ha sido utilizada en el análisis de sistemas biológicos, como puede ser la relación de dependencia en la activación de neuronas en el cerebro en la realización de

actividades conscientes [46]. En el caso de este trabajo, la información mutua entre entre la señal emitida y el impulso recibido por el sistema biológico nos permite caracterizar la influencia del estimulo sobre el estado del pez.

En resumen, puesto que partimos del desconocimiento sobre el modo en que los peces de la especie *Gnathonemus Petersii* codifican la información a transmitir en la señal emitida, un estudio estadístico basado en teoría de la información sobre cómo tiene lugar dicho procesamiento en dicho sistema biológico puede resultar muy revelador.

1.2.1. Influencia del retraso pulso-estímulo

Entendemos por retraso pulso-estímulo el tiempo transcurrido entre el último pulso emitido por el pez previo al lanzamiento de un estímulo artificial y el lanzamiento efectivo de dicho estímulo. Puede observarse de manera gráfica en la figura 1.6

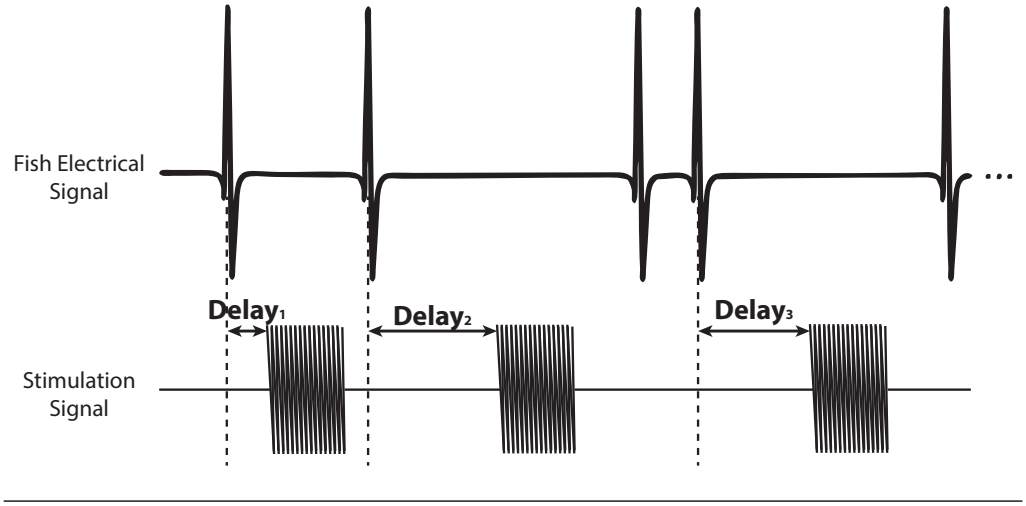

FIGURA 1.6: Inter-Pulse Interval.

La influencia de este retraso pulso-estímulo en la estimulación dependiente de actividad del pez, a la hora de estudiar el procesamiento de información de este organiso, ya ha sido analizada en [47]. Este artículo muestra cómo existe un tiempo de retraso crítico en la interacción en ciclo cerrado entre la presencia de un pulso del pez y la estimulación; ya que los cambios en el comportamiento eléctrico son mucho más apreciables cuando el retraso está por debajo de los 100ms.

Resulta necesario tener en cuenta este resultado a la hora de establecer un ciclo cerrado de estimulación eléctrica con un organismo electrorreceptivo. Para lograr un protocolo útil y efectivo de cara al estudio la estimulación dependiente de actividad ha de ser captada por el pez de la manera más precisa posible, con el objetivo de que las dinámicas de procesamiento de información que nos interesa estudiar salgan a la luz.

1.2.2. Influencia de la memoria a corto plazo

En el caso que nos ocupa, el estudio de la señal del *Gnathonemus petersii*, el evento diferenciado en la señal eléctrica son los pulsos. Como ya se ha comentado, la temporalidad de los pulsos tiene una importancia clave en la codificación de información en la señal. Resultados recientes en el campo del análisis del procesamiento de información en peces eléctricos muestran una respuesta del pez a estimulación dependiente de eventos instantaneos como un pulso del pez [12]. Sin embargo, es de esperar que la respuesta varíe cuando la estimulación se realice en función de eventos que tengan en cuenta lo ocurrido no solo en el instante inmediantamente anterior, si no en un rango de tiempo más elevado. Dicha estimulación estaría teniendo en cuenta la influencia de la memoria a corto plazo del organismo, de manera que sería posible estudiar su influencia en los procesos de comunicación. La influencia de la memoria en los procesos electrorreceptivos es un campo poco estudiado a pesar de que tiene una importancia clave, tanto para la comunicación como para la electrorrecepción. En el caso comunicativo la existencia de memoria es lo que permitiría una comunicación bidireccional que vaya más allá de la respuesta a eventos instantaneos, mientras que en el caso de la electrolocalización, la memoria a corto plazo es la que permitiria detectar el movimiento de un objeto y no únicamente su posición estática.

Sería posible analizar la influencia de una memoria a corto plazo en el procesamiento de información y la generación de señal eléctrica del pez tomando en cuenta un intervalo mayor de la señal a la hora de guiar la estimulación. De tal modo y con el objetivo de ir más allá de la respuesta a eventos instantaneos, en este proyecto se ha decidido estudiar la influencia de una memoria a corto plazo en los procesos electroreceptivos. Para tomar en cuenta esta posibilidad se tomarán, como sucesos portadores de información en la señal, la generación de determinados códigos temporales o palabras y se hará uso de una estimulación dependiente de la emisión de estos códigos por parte del pez.

La definición de dichos códigos se hará en función de los sucesos instantaneos (los pulsos) en intervalos de tiempo. La presencia de un pulso en el intervalo se codificará como '1' y la ausencia de pulso, como '0', de tal manera que estamos digitalizando y binarizando la señal eléctrica adquirida.

Tomando varios de estos intervalos generamos un código binario o palabra. La selección del tiempo de intervalo y del tamaño de palabra es una decisión fundamental del proceso [...] Se ha seguido un criterio de máxima entropía para tratar de maximizar la capacidad de transmisión de información de la señal resultante del proceso de digitalización.

1.3. Técnicas de estimulación en ciclo cerrado

La estimulación en ciclo cerrado es una técnica que consiste en presentar al sistema bajo estudio una estimulación que dependa de la propia actividad del sistema observado. El ciclo de estímulo-respuesta se establece mediante la monitorización de la actividad biológica usando distintos tipos de sensores y la utilización de un algoritmo que actualiza sus parámetros de manera on-line, lo que le permite detectar determinados eventos e iniciar o detener la estimulación en base a esa detección. De este modo se da una interacción bidireccional entre el sistema biológico y la estimulación artificial. Puede verse un esquema de este modelo en la figura 1.7.

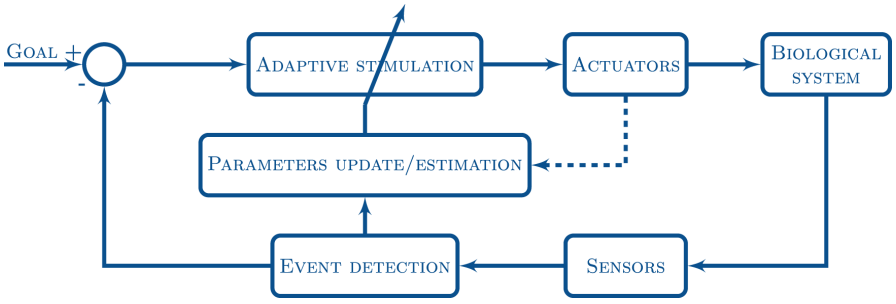

Figura 1.7: Representación esquemática de una generalización del ciclo cerrado para la estimulación dependiente de actividad. Extraido de [1].

La motivación de utilizar este tipo de técnicas en ciclo cerrado se debe a la posibilidad de revelar en los sistemas estudiados dinámicas que permanecen ocultas bajo protocolos tradicionales de estimulación unidireccional, como podrían ser las referentes a los mecanismos de comunicación, memoria y aprendizaje [1, 48]. Por último, estos procedimientos pueden ser utilizados como técnicas de control sobre el estado del sistema.

El uso de este tipo de técnicas de estimulación closed-loop se viene aplicando desde los años 40 en neurociencia y en experimentos de electrofisiología bajo el concepto de *dynamic clamp* [40, 41] y siguen siendo utilizadas en la actualidad [48, 49]. Dicho concepto consistía originalmente en la inyección de corrientes en neuronas vivas en función del potencial de membrana de las mismas, lo que permitía construir redes híbridas para estudiar la transmisión de información entre neuronas de la red [40, 41]. Es posible generalizar el concepto de *dynamic clamp* para diferentes contextos experimentales en neurociencia que requieren de protocolos de estimulación bidireccional [48, 50–52]. Por ejemplo, la estimulación mecánica dependiente de la actividad en ráfagas en el sistema de navegación del *Clione limacina*[1] o la interfaz cerebro-máquina teniendo en cuenta la variabilidad entre individuos para optimizar diversos protocolos[52].

Por tanto, el uso de estas técnicas de estimulación en ciclo cerrado operando en tiempo real permite la adquisición de datos y la estimulación dependiente de actividad. En el

contexto de este proyecto, la generalización de dicho concepto permite realizar experimentos que analicen la actividad del sistema biológico en relación con su ecosistema, ya que es posible replicar la interacción bidireccional propia de los procesos comunicativos que tienen lugar de forma natural. Esto hace posible una interacción optima y controlada entre los dispositivos artificiales y los sistemas de procesamiento de información naturales del pez. Este modelo en ciclo cerrado resulta fundamental para caracterizar y extraer información del sistema en vivo y, además, el ciclo cerrado puede establecerse de forma no-invasiva, lo que permite un control de la actividad durante períodos de tiempo muy largos. Este tipo de técnicas se han utilizado de manera más concreta también al estudio de la electrorecepción activa [12, 47]. Un esquema teórico de cómo puede aplicarse un ciclo cerrado al estudio del pez eléctrico se encuentra en la figura 1.8

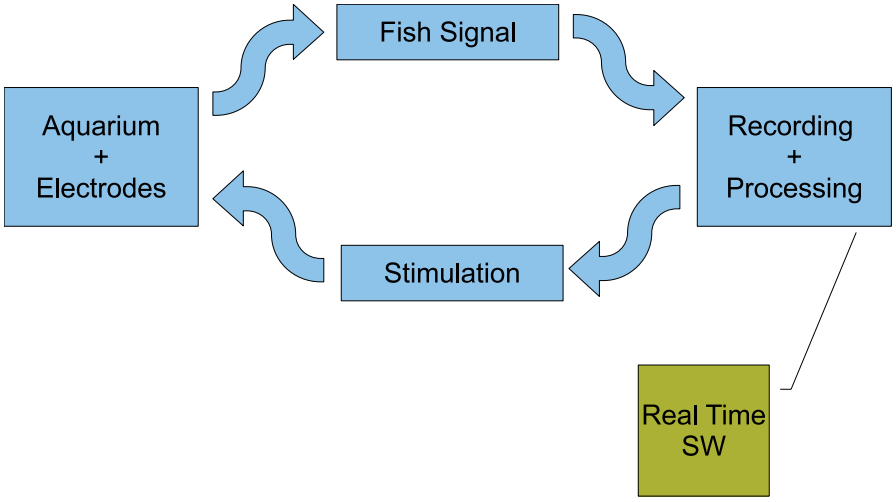

FIGURA 1.8: [Representación esquemática de la aplicación de un ciclo cerrado al estudio del pez eléctrico

1.3.1. Sistemas en tiempo real

Para el correcto funcionamiento del ciclo cerrado resulta necesario el uso de tecnologías en tiempo real.

Un sistema en tiempo real (STR) se define como un sistema que realiza una interacción con un proceso físico del mundo real y que emite respuestas correctas cumpliendo las restricciones temporales que se le imponen. Esto significa que el sistema debe disponer de la capacidad de responder a eventos de manera prácticamente instantánea y temporalmente predecible ya que retardos inesperados en la respuesta del sistema pueden tener un alto coste.

Las características de un STR son, por tanto:

- Capacidad de determinar con precisión el tiempo de una tarea en iniciarse.
- Responsividad respecto a:
  - La cantidad de tiempo lleva el iniciar la ejecución de una interrupción.
  - La cantidad de tiempo que se necesita para realizar la tarea que pidió la interrupción.
- Estabilidad del sistema respecto a errores (y preservación de los datos).

Habitualmente se dividen las aplicaciones que funcionan en tiempo real en dos subgrupos, soft real-time (o tiempo real acrítico) y hard real-time (tiempo real crítico). En este último grupo se incluyen aquellas aplicaciones que deben cumplir de manera más estricta los requerimientos temporales, de tal modo que las latencias deben mantenerse bajas y bien delimitadas, mientras que las aplicaciones soft real-time tienen cierta tolerancia a los requerimientos temporales de respuesta. Un ejemplo de hard real-time podría ser el airbag de un coche, si este estuviese controlado por software. Un ejemplo de una aplicación soft real-time podría ser el streaming de audio o video por internet, donde el retardo o la pérdida de algunos paquetes de datos no resulta crucial para el desempeño general de la aplicación.

El uso de tecnologías en tiempo real resulta esencial para el estudio de sistemas biológicos si queremos establecer de manera efectiva un ciclo cerrado de estimulación bidireccional. Por ejemplo, para el estudio de la codificación sensorial en el sistema nervioso de organismos vivos podemos utilizar una estimulación mecánica, pero puesto que para producir estímulos mecánicos que resulten realistas se requiere de una respuesta en el orden temporal de los milisegundos, hay que hacer uso de tecnologías de control en tiempo real [53]. En el caso que nos ocupa, el orden temporal en el que trabaja el pez también es de milisegundos, ya que la anchura aproximada del pulso es de 1ms. [9].

1.4. Resumen de objetivos

Son objetivos principales de este proyecto:

1. Estudiar el modo en que la información se procesa en la señal electrorreceptiva de los peces eléctricos con la intención de entender y caracterizar mejor los procesos de electrolocalización y electrorrecepción, con importantes aplicaciones prácticas en otros campos, como la robótica. Como objetivos parciales a este:

   a) Evaluar la influencia de la memoria en el procesamiento de información

1) Desarrollo de mecanismos de estimulación en ciclo cerrado guiados por códigos que portan información sobre la señal del pez en el rango de tiempo inmediatamente anterior.
2) Implementación de un protocolo de digitalización y binarización de la señal.
3) Selección de unos parámetros del protocolo de binarización correctos para cada especimen
4) Desarrollo de protocolos de estimulación adecuados al estudio de la influencia de la memoria, que tengan en cuenta las características de la especie y el especimen concreto.

b) Analizar la capacidad de transmisión de información de la señal eléctrica y la evolución temporal de dicha capacidad.

1) Cálculo de la entropía o capacidad de transmisión de información de la señal emitida por el pez.
2) Herramientas de análisis de su evolución a lo largo del tiempo.

c) Considerar la importancia del retraso post-pulso en la recepción de estímulos para el estudio de los cambios en el procesamiento de información.

1) Aplicación al problema de tecnologías de tiempo real para asegurar la precisión temporal del retraso

2. Desarrollo de metodologías de análisis del procesamiento de información que puedan aplicarse al estudio de otros sistemas biológicos.

3. Implementación de protocolos de estimulación en ciclo cerrado que puedan aplicarse al estudio de otros sistemas biológicos.

4. Desarrollo de aplicaciones en tiempo real para su aplicación al estudio de otros sistemas biológicos.

Capítulo 2

Metodología

2.1. Sistema biológico *Gnathonemus Petersii*

Trabajamos con individuos de la especie *Gnathonemus Petersii*. Dicha especie es originaria del África Occidental y sus características pueden consultarse en la Introducción (Sección 1).

En la realización de los experimentos se han utilizado 3 especímenes de entre 12cm y 14cm, a los que hemos decidido numerar del 1 al 3.

Cuadro 2.1: Tabla de especímenes experimentales

Número	Longitud	Periodo experimental
1	13 cm	09/05/2014 a 23/06/2014
2	12,5 cm	09/05/2014 a 16/07/2014
3	14 cm	09/09/2014 en adelante

Los especímenes fruto de estudio fueron obtenidos en tiendas de animales de Madrid, España. Se les mantuvo en los acuarios utilizados para los experimentos de manera individual y sin restricciones a su movimiento.

Los acuarios utilizados para mantener a los peces han sido dos, a los que hemos decidido nombrar como A y B. El acuario A tiene un volumen de 65 litros de agua y unas dimensiones de 80cm x 30cm x 25cm; dicho acuario cuenta con una barrera física a la mitad del mismo (lo que deja unas dimensiones de 40cm x 30cm x 25cm) que permite el paso de agua pero no el paso del pez. El acuario B tiene un volumen de 20 litros y unas dimensiones de 45cm x 22cm x 20cm. Dichos acuarios se han visto expuestos a iluminación artificial, con una temperatura de controlada de 25^{o}C aproximadamente, con un suelo de grava especial para acuarios y sin plantas. El control de la temperatura

se realiza mediante un calentador. También contamos con un filtro EHEIM Airpump 100 que mantiene la calidad del agua y la mantiene libre de residuos orgánicos o químicos.

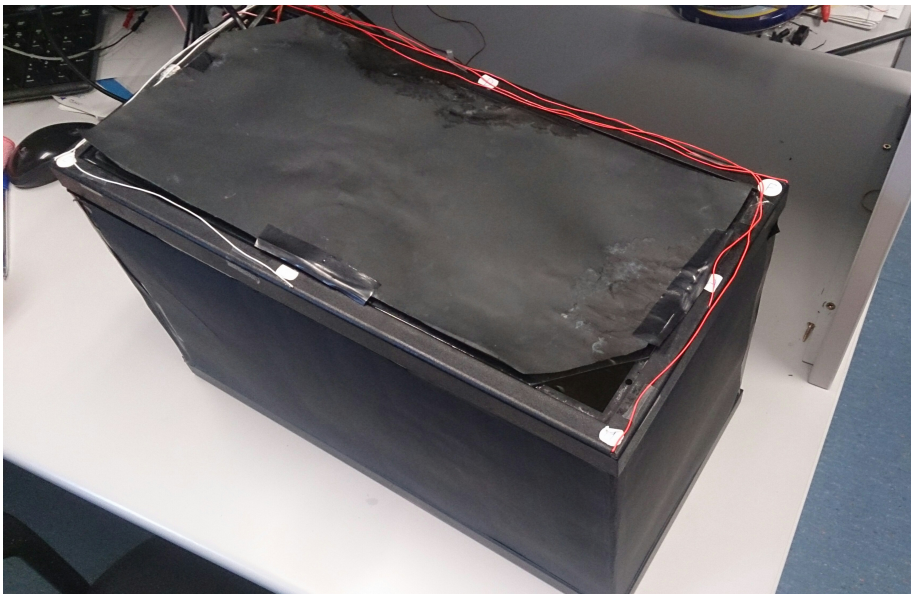

FIGURA 2.1: Imagen del acuario A

FIGURA 2.2: Imagen del acuario B

Para conseguir un ambiente más cómodo para el propio animal se ha impedido la entrada de un exceso de luz mediante la colocación de unas cartulinas oscuras en las paredes

acristaladas del acuario. Tras la colocación, se observó que los especímenes hacian uso de todo el espacio disponible en el acuario en lugar de mantenerse en los espacios más oscuros del mismo.

Estos acuarios han sido utilizados no solo para mantener a los peces, sino en la realización de experimentos, por lo que puede encontrarse más información sobre los mismos en la sección 2.3, que trata sobre la arquitectura hardware.

2.2. Técnicas de análisis de procesamiento de información

2.2.1. Intervalo entre pulsos

Para la realización del análisis que se detalla en este trabajo se ha seleccionado un modelo de procesamiento de la información en la señal del pez basado en el tiempo entre pulsos que denominamos IPI, del inglés *Inter-Pulse Interval* (Figura 1.5). Puede observarse una descripción gráfica de la magnitud que representa un IPI en la figura 1.5. El comportamiento eléctrico y cómo este se ve modificado en relación a cambios en el entorno o a bajo condiciones experimentales distintas puede caracterizarse de manera simple en base a la construcción de histogramas de IPIs. Estos histogramas nos permiten observar de manera gráfica cambios en el comportamiento electrico del pez, ya que muestran la probabilidad de ocurrencia de un determinado intervalo entre pulsos. Este tipo de histogramas han sido utilizados de manera efectiva en estudios previos [10–12]. Puede verse un ejemplo de este tipo de histogramas en la figura 2.3.

Además de estos histogramas de IPIs, podemos utilizar lo que se conoce como gráficos cuantil cuantil (o q-q plots) para comparar las distribuciones de IPIs bajo condiciones experimentales distintas en un mismo experimento. Estos gráficos permiten comparar dos distribuciones de IPIs ya que son una herramienta concebida para comparar las distribuciones de dos o más conjuntos de datos, de tal modo que permite detectar variaciones entre ambas.

Los cuantiles son un concepto matemático que se aplica sobre las distribuciones. El cuantil n, q(n), de un conjunto de datos es el valor que cumple que una fracción f de los datos son menores o iguales a q(f). Es decir, el cuantil 0.5 es el valor que deja por debajo al 50 % de los datos, lo que se puede denominar también como el percentil 50, segundo cuartil o la mediana. De tal modo, el cuantil 0.25 se le llama cuartil inferior, el cuantil 0.5 es el cuartil medio y el cuantil 0.75 es el cuartil superior.

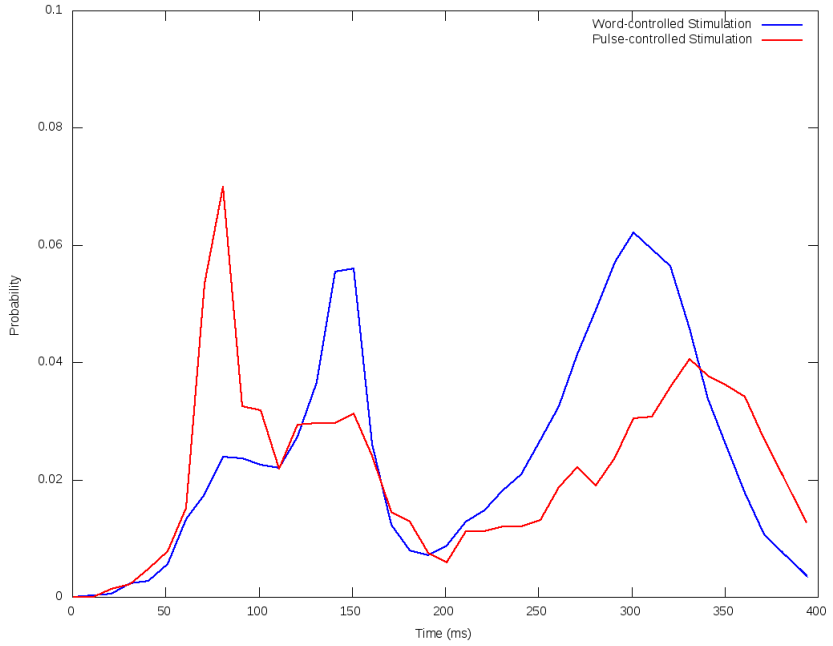

FIGURA 2.3: Histograma de IPIs

Para realizar los gráficos cuantil-cuantil se procede del siguiente modo: Suponiendo que existan 2 conjuntos de medidas a ser comparadas, donde

$$x(1), x(2), ..., x(n)$$

es el primer conjunto ordenado en orden creciente e

$$y(1), y(2), ..., y(m)$$

es el segundo conjunto también ordenado en orden creciente. Suponiendo $m \leq n$.

Si $m = n$ entonces x(i) e y(i) entonces son ambos $\frac{i-0,5}{n}$ cuantiles de sus respectivos conjuntos de datos y construimos el gráfico según el valor de x(i) frente al de y(i). Es decir, que los puntos P(i) del gráfico vienen dados por $P(i) = (x(i), y(i))$. Si $m < n$ entonces construimos el gráfico de y(i) contra el $\frac{i-0,5}{m}$ cuantil de los datos x, que puede ser calculado mediante una interpolación.

Podemos ver un ejemplo de grafico q-q plot en la figura 2.4.

Para analizar estos gráficos q-q plot tomamos como referencia una linea recta a 45º. Si ambas distribuciones son idénticas, los datos se ajustarán a esta referencia. Si los

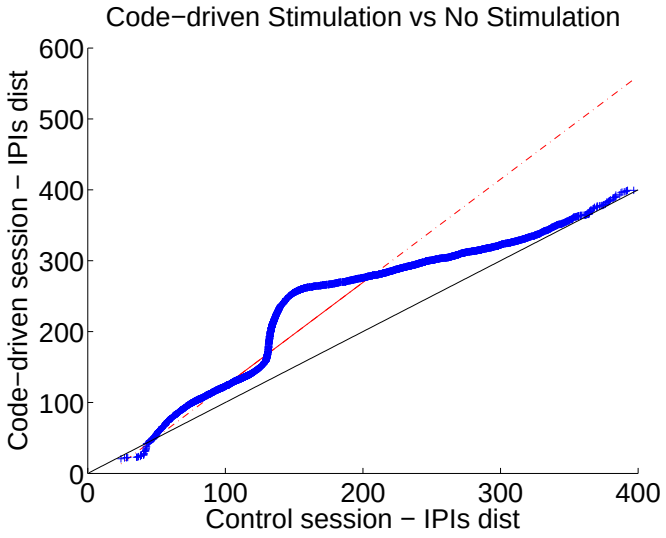

FIGURA 2.4: Gráfico q-q plot de 2 conjuntos X e Y.

conjuntos de datos tienen distribuciones que solo difieren en la posición, los puntos estarán alineados con esta recta a 45º pero desplazados hacia arriba o abajo.

Por otro lado, la recta discontinua que observamos representa el rango intercuartílico y tiene este nombre porque se forma uniendo el primer y el tercer cuartil de los datos e interpolando linealmente. Si los datos se ajustan a esta recta, significa que la distribución subyacente entre los dos conjuntos concuerda y que los datos de una distribución son solo una transformación lineal de los de la otra. De ese modo el análisis del q-q plot no se ve modificado por una transformación lineal de los datos. Si la pendiente de la recta intercuantílica es menor de 45º, la distribución representada en el eje horizontal tiene una dispersión mayor que la distribución representada en el eje vertical. Al contrario, si la pendiente es más pronunciada que la referencia, la distribución representada en el eje vertical es más dispersa que la distribución del eje horizontal. Los datos a menudo se arquean, o tienen forma de "S", lo que indica que una de las distribuciones está más sesgada que la otra, o que una de las distribuciones tiene colas más pesadas.

En el caso concreto de nuestros gráficos q-q plot utilizados para comparar las distribuciones de IPIs bajo distintas condiciones experimentales, si los puntos caen exactamente sobre la linea de referencia, signica que la estimulación en ciclo cerrado no afecta al pez. Si los puntos recaen sobre una línea con una pendiente de 45º situada por encima de la referencia, los IPIs durante la estimulación son más largos (menor frecuencia) que durante la sesión de control. A la inversa, si la línea se sitúa por debajo, los IPIs durante la estimulación son más cortos (mayor frecuencia) que durante el control. Finalmente, cuando los puntos recaen sobre una recta con pendiente distinta de 1, los puntos por encima de la referencia son aquellos donde los IPIs de la sesión en ciclo cerrado son más

largos y los puntos por debajo de la referencia son aquellos en los que los IPIs de la estimulación son más cortos. Esto representa un cambio en la dispersión de la distribución de IPIs, por lo que si los puntos se alejan de la referencia podemos afirmar que hay un cambio en la distribución subyacente de emisión de IPIs y que, por tanto, la estimulación en ciclo cerrado ha afectado al pez modificando a su procesamiento de información.

2.2.2. Codificación binaria de la señal electroreceptiva

Como ya se ha explicado en la introducción (sección 1.2.2), para el estudio de la influencia de la memoria a corto plazo en el procesamiento de información en los mecanismos electrorreceptivos, hemos hecho uso de una digitalización binaria de la señal original del pez y, posteriormente, hemos particionado la señal en códigos binarios de un número de bits.

La digitalización de la señal tiene lugar del siguiente modo: Pueden definirse los códigos en función de los sucesos instantaneos (los pulsos) que tienen lugar o no en un cierto intervalo de tiempo. Es decir, codificamos la existencia o no de un pulso del pez en intervalos cortos de tiempo. La existencia de pulso en un intervalo de tiempo se codifica con el símbolo '1' y la inexistencia de pulso con el símbolo '0', como si se tratase de un bit. La longitud temporal de dichos intervalos la hemos denominado como tiempo de bin (*Bin time* en inglés) o Δt. Tomamos varios de estos intervalos para generar un código binario o palabra, el número de intervalos se denomina longitud o tamaño de palabra (o wl por *Word length*, en inglés). Un ejemplo de este proceso puede observarse en la figura 2.5.

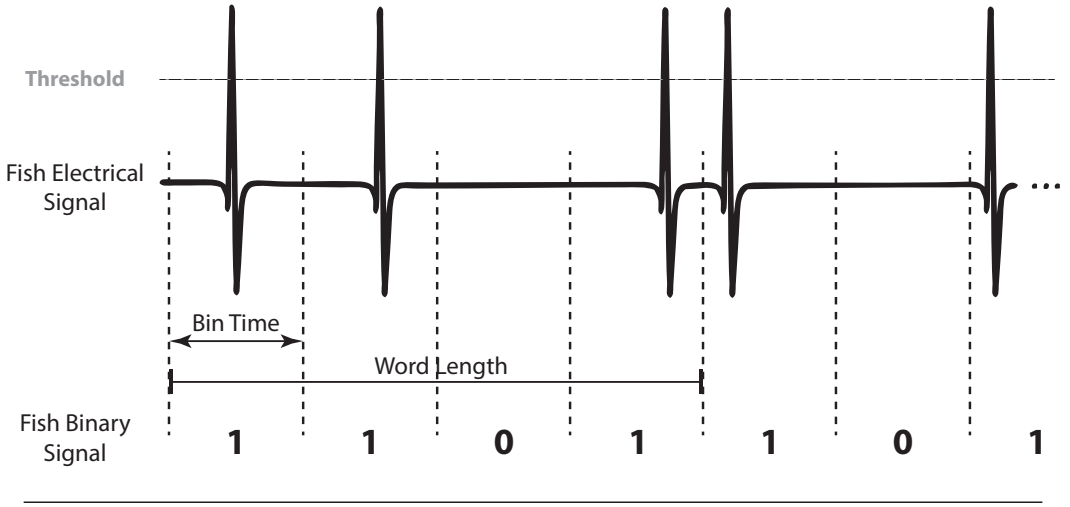

FIGURA 2.5: Binarización de la señal

La partición de la señal binaria en códigos se realiza con sobreposición. Esto quiere decir que tomamos una ventana de n bits (donde n corresponde al tamaño de la palabra) para

detectar la palabra i-ésima y, para detectar la palabra i+1, se realiza un desplazamiento de un solo bit. De tal modo que el segundo bit de la palabra i-ésima es el primero de la palabra i+1. Puede verse un ejemplo en la figura 2.6

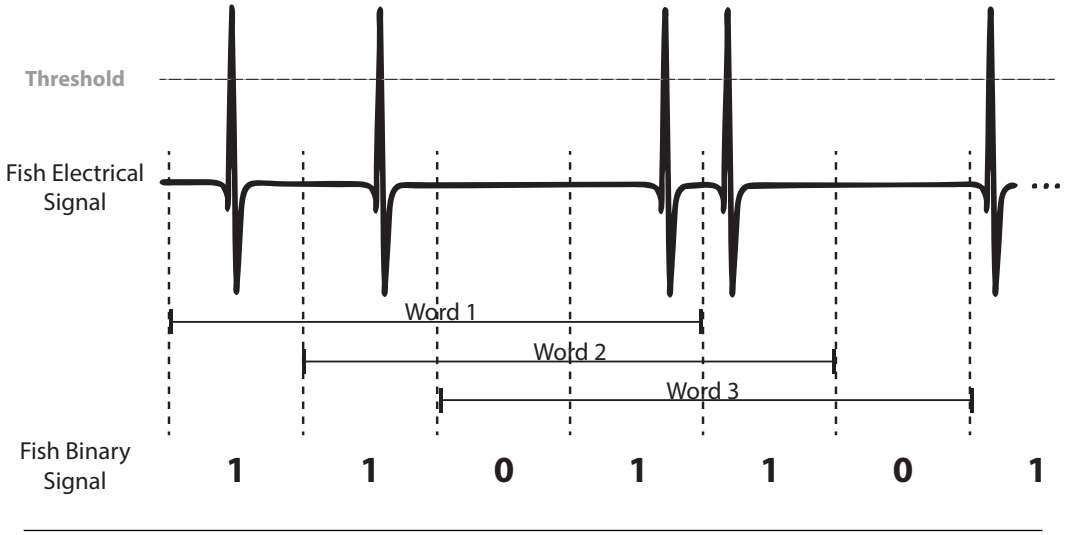

FIGURA 2.6: Codificación de la señal con desplazamiento de 1 bit.

La selección de los parámetros tiempo de bin y tamaño de palabra del modelo de codificación seleccionado se realiza de manera experimental aplicando un criterio de máxima entropía. De tal modo, tras almacenar la señal del pez en condiciones normales sin estimulación, se generan los histogramas de códigos emitidos en función del tiempo de bin y del tamaño de palabra. Un ejemplo de estos histogramas puede verse en la figura 2.7.

Las distribuciones se generan en función de los dos parámetros especificados: tiempo de bin y tamaño de palabra. Los tiempos de bin analizados son de 20ms a 150ms en intervalos de 10ms. Respecto a los tamaños de palabra, se han analizado de 2 a 5 bits. A partir de dichos histogramas se genera la distribución de probabilidad de los códigos que nos permite calcular la entropía de cada una de las distribuciones. La entropia se calcula mediante la ecuación 1.2, donde cada evento X es uno de los posibles códigos y su probabilidad viene dada de manera empírica por el número de apariciones sobre el total. Así podemos representar una superficie entrópica como la de la figura 2.8 que nos permite seleccionar, mediante un criterio de máxima entropía, los parámetros de la codificación.

Es importante notar que la entropía aumenta con el número de estados, es decir, con el número de posibles códigos, ya que se calcula como un sumatorio sobre el total de estados. El número de estados depende del número de bits, ya que con n bits obtenemos 2^n códigos posibles. Por tanto, se ha decidido normalizar la entropía dividiendo el resultado por el número de bits. A esta magnitud le hemos llamado entropía por bit y es la que viene representada en la figura 2.8.

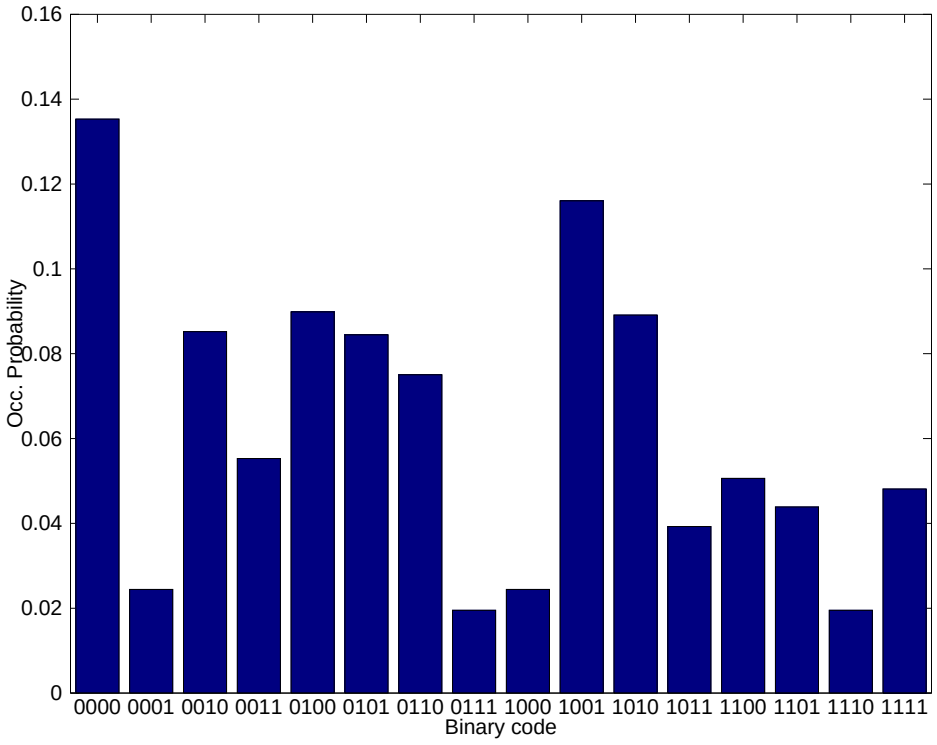

FIGURA 2.7: Histograma de códigos binarios. Este histograma es el resultado de aplicar el modelo de codificación a la señal almacenada en un experimento. Muestra la distribución de probabilidad de los distintos códigos binarios.

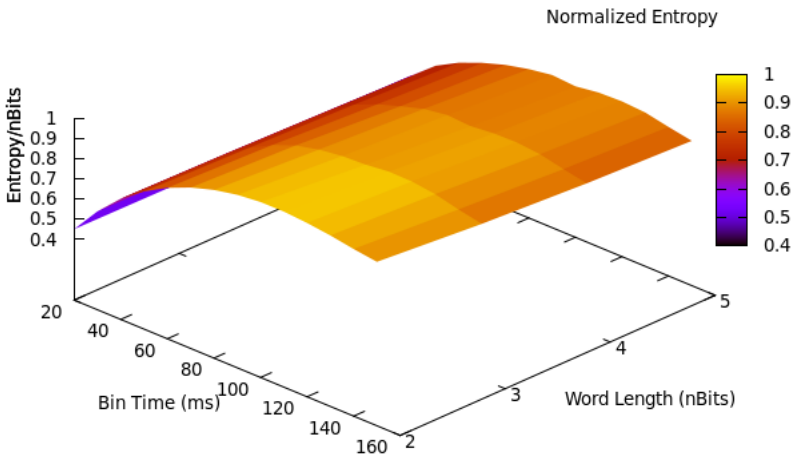

FIGURA 2.8: Superficie entrópica en función de los parámetros de codificación.

Este proceso puede ser realizado de manera previa y utilizado para una serie de experimentos o bien previo a cualquier experimento. Para analizar los cambios en el comportamiento entrópico y determinar si este resulta muy variable temporalmente o en función del estado del pez (en cuyo caso sería conveniente un análisis previo a cada experimento y no un análisis único para cada serie) se han realizado los experimentos sobre la evolución de la distribución de códigos emitidos y de la entropía en sistemas sin estimulación. Los resultados de dichos experimentos pueden encontrarse en la sección 3.1.

2.2.3. Protocolos de estimulación

El protocolo de estimulación basado en palabras binarias consiste en la recepción de la señal electrorreceptiva emitida por el pez eléctrico, la binarización de dicha señal tal como se explica en la sección 2.2.2 y el envío de estímulos dependiente (o no) de dicha señal.

En concreto hemos desarrollado dos protocolos, uno que establece un ciclo cerrado de estimulación con el sistema biológico y otro que pretende simular determinadas propiedades del ciclo cerrado sin establecerlo efectivamente.

El protocolo de estimulación en ciclo cerrado dirigido por códigos actúa del siguiente modo: Toma el voltaje de la señal a cada instante y realiza una binarización basándose en los parámetros suministrados mediante la interfaz de usuario, para ello necesita detectar los pulsos del pez mediante el uso de un voltaje umbral. Cuando la señal cruza dicho voltaje umbral se detecta un pulso y se establece el bit correspondiente a la ventana de tiempo activa a 1. Si la ventana de tiempo termina sin detectar pulso, el bit correspondiente se establece a 0. A partir de dicha binarización realiza un cálculo de la palabra activa y la compara con la palabra a detectar. Si coinciden, activa la detección de palabra, almacenando el tiempo en que se ha detectado. A partir de ahí seguiremos detectando pulsos y una vez se cumple el rango de tiempo de retraso entre pulso estímulo suministrado como parámetro, lanzaremos la estimulación. Dependiendo del experimento el protocolo de estimulación utiliza un retraso pulso-estímulo fijo o dentro de un rango variable. Respecto a la estimulación aleatoria, el protocolo trata de mantener las siguientes propiedades de la estimulación en ciclo cerrado:

- Tipo de estímulo presentado
- Densidad de estímulos por unidad de tiempo
- Número total de estímulos
- Retraso pulso-estímulo

debido a ello, debemos conocer el número de estímulos total y la cantidad de tiempo empleado en utilizarlo en el caso de la estimulación en ciclo cerrado. El tipo de estímulo a utilizar en el caso aleatorio será el mismo. Debemos suministrar como parámetro el periodo de estimulación necesario para replicar la densidad de estímulos por unidad de tiempo. De tal modo, el protocolo espera un tiempo aleatorio dentro de ese periodo antes de lanzar el estímulo, una vez se cumple dicho tiempo, se espera la detección de un pico y, a partir de él, el retraso necesario para estimular de un modo que replique el modelo en ciclo cerrado. Se han utilizado modelos de retraso pulso-estímulo fijo y variable dentro de un rango en los distintos experimentos realizados, tal como se explica en el capítulo 3.

El protocolo de estimulación en ciclo cerrado sigue el esquema presentado en la figura 2.9. Mostramos el funcionamiento del protocolo de estimulación aleatorio en la figura 2.10.

La decisión sobre estimular o no en la decisión *¿Hay que estimular?* depende de que se haya cumplido el tiempo del retraso con respecto al último pulso registrado. En los experimentos preliminares, tal como se explica en la sección 3.2, los retrasos eran variables dentro de un rango determinado. Posteriormente, se realizó un estudio que consistía en comparar el procesamiento de información derivado de una estimulación basada en códigos con un retraso fijo de 10ms. respecto de una estimulación con un retraso en rango. Dichos experimentos pueden observarse en la sección 3.3. Como actualización derivada de dicho estudio realizado y el análisis de sus resultados, se decidió que en los experimentos sobre memoria que listamos en la sección 3.4 ambos protocolos de estimulación enviasen los estímulos con un retraso fijo de 10ms.

La descripción del módulo software encargado de hacer uso de estos protocolos se encuentra descrito en la sección 2.6.1. El código C que implementa los protocolos seleccionados se encuentra replicado en el apéndice B.

2.3. Arquitectura Hardware

La arquitectura hardware para la realización de los experimentos cuenta con un montaje de electrodos para la detección en el acuario, un circuito amplificador-sumador que recibe las señales captadas por los dipolos de la pecera y genera una señal única a partir de ellas, una tarjeta de adquisición de datos (DAQ) que recibe la señal analógica y la transmite al PC. Finalmente, contamos también con un dispositivo para la emisión de estímulos que simula un pez eléctrico artificial.

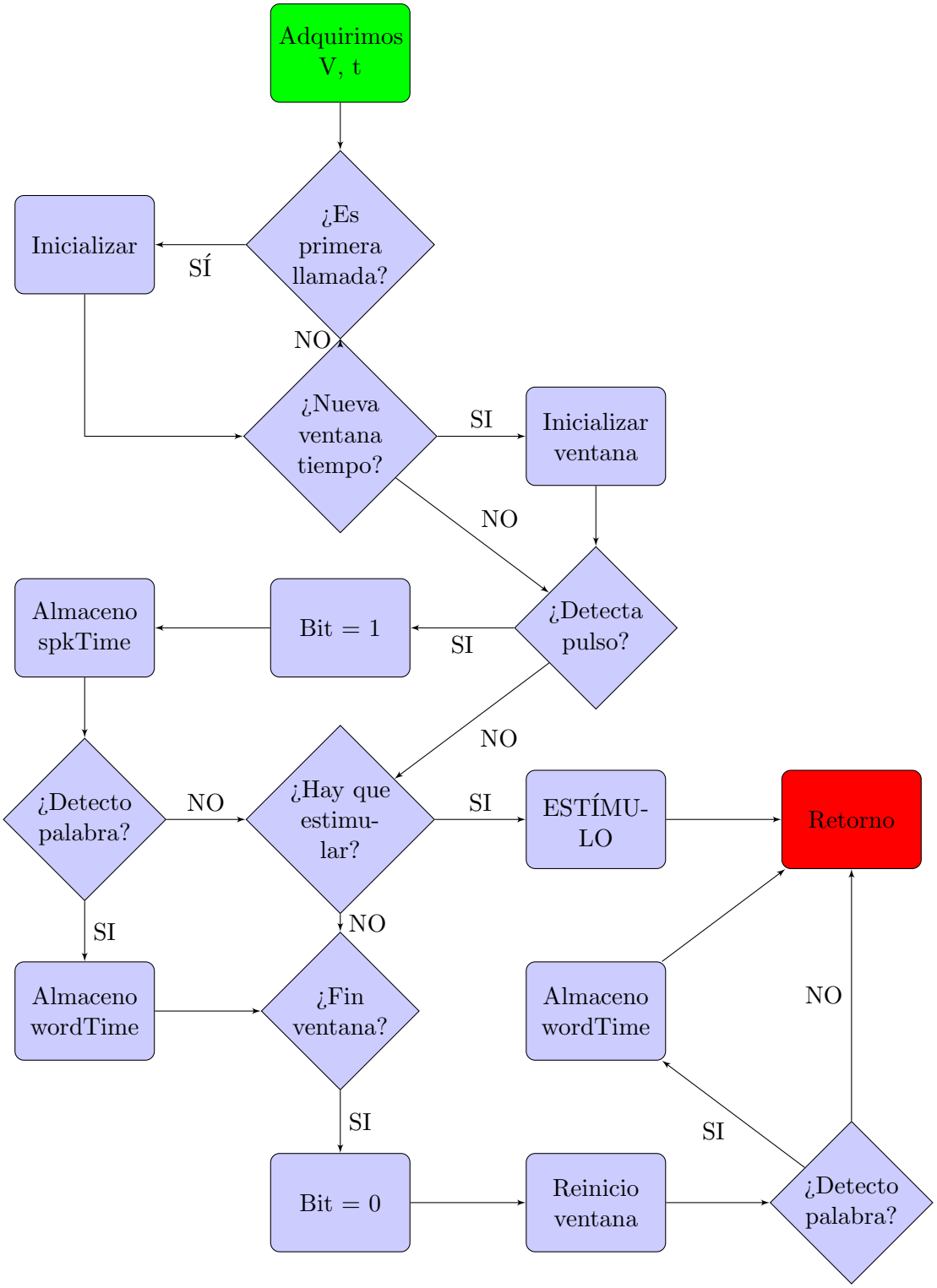

FIGURA 2.9: Diagrama del protocolo de estimulación en ciclo cerrado guiado por códigos.

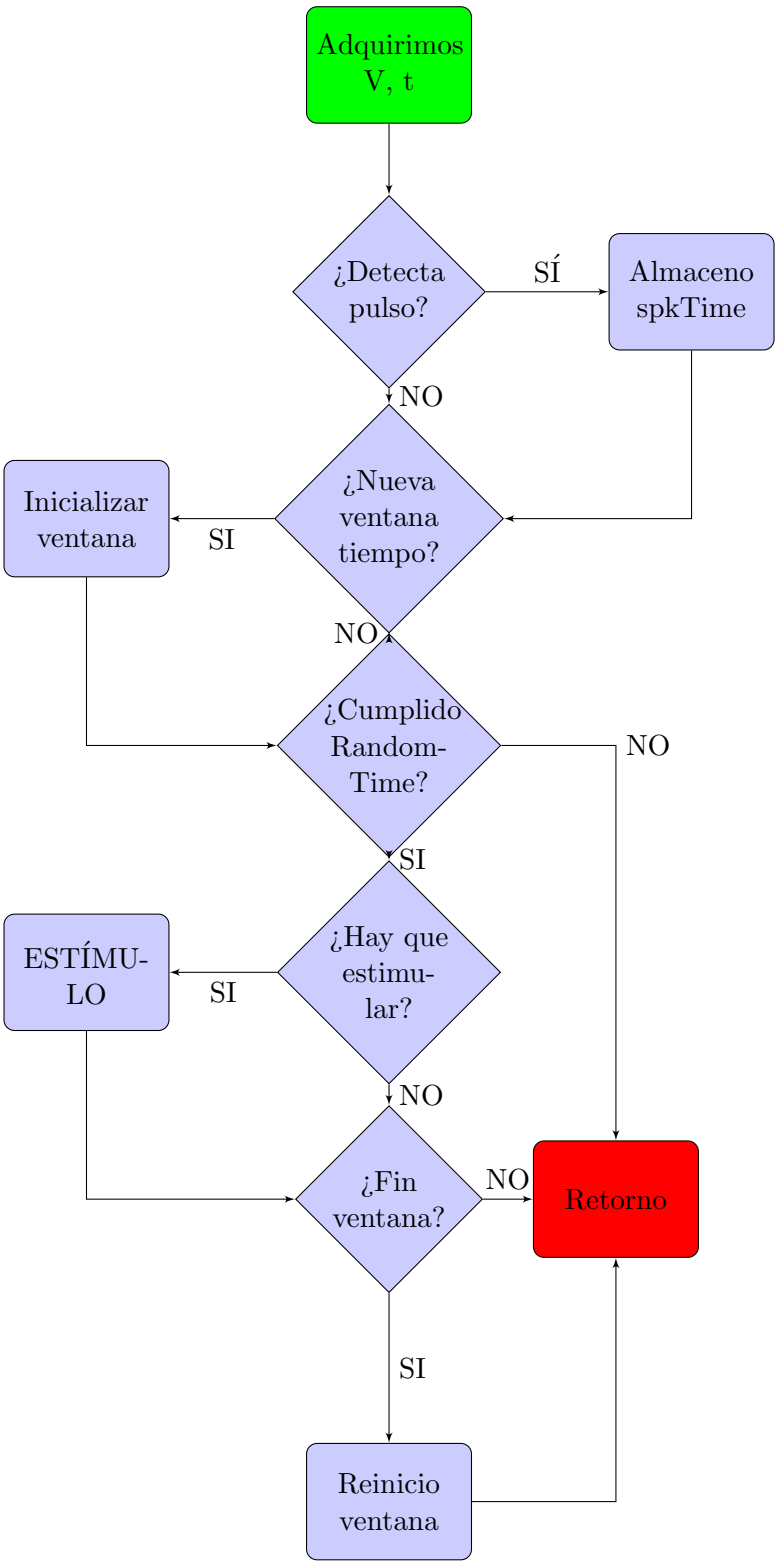

FIGURA 2.10: Diagrama del protocolo de estimulación aleatorio.

Una representación esquemática del montaje utilizado para la realización de los experimentos se encuentra en la figura 2.11.

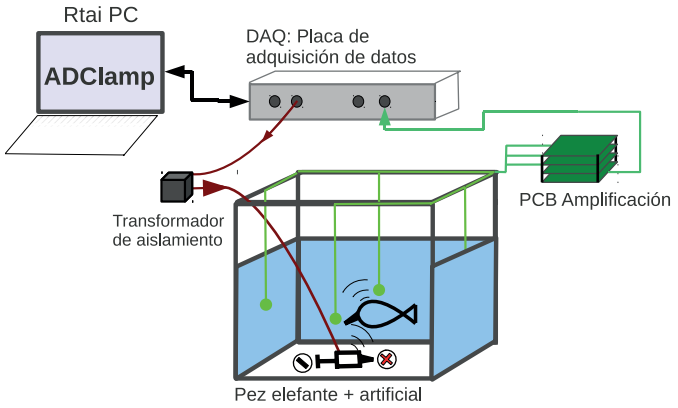

FIGURA 2.11: Dos setups distintos para la disposición de dipolos para la adquisición de la señal eléctrica del pez en el acuario. A la izquierda el setup A, del que partimos en la investigación. A la derecha el desarrollado en el GNB durante la investigación y que hemos denominado setup B.

2.3.1. Detección de la señal eléctrica

La manera más simple de medir la señal eléctrica generada por los peces es a través de dos electrodos amplificados diferencialmente, lo que conocemos como dipolo. Bastaría con colocar uno de los electrodos cerca de la cabeza del especimen y el otro próximo a la aleta caudal. El problema principal de este método, como se puede intuir, es mantener al pez en una posición tal que los electrodos queden debidamente colocados. En un acuario donde el pez pueda nadar libremente, cuando el pez se aleja de los electrodos la amplitud de voltaje detectada en el dipolo se reduce y dificulta (o, incluso, imposibilita) la detección de los pulsos eléctricos. También dependiendo de la orientación del pez la amplitud puede variar considerablemente, por ejemplo, si imaginamos el pez situado en el centro del dipolo y orientado en perpendicular al mismo, la amplitud detectada en ambos electrodos va a ser teóricamente equivalente, por lo que el diferencial se acerca a cero.

Debido a que queremos medir peces nadando libremente en el acuario, se ha establecido un setup consistente en múltiples dipolos repartidos en los extremos del acuario para la adquisición de la señal del pez. Para una correcta recepción de los campos eléctricos generados por el pez es necesario tener en cuenta la conductividad electrica del metal receptor. Se han utilizado puntas metálicas bañadas en plata.

A lo largo del desarrollo de este trabajo se han utilizado dos montajes distintos respecto a la distribución de dipolos en los acuarios. Hemos nombrado a estos dos setups como

distribución A y distribución B. La distribución A es una versión inicial, mientras que la distribución B es una mejora de la misma que evita problemas de recepción de la señal debidos a la posición del pez y ha sido realizada en el GNB al mismo tiempo en que tenía lugar este proyecto [54]. Puede observarse un diagrama de los dos setups utilizados en la figura 2.12. Como mejora adicional, en el montaje del setup B se han colocado los dipolos a una altura media entre el suelo del acuario y la superficie. En el setup A, los dipolos de detección se distribuían sobre el fondo del acuario.

En el acuario A se ha utilizado el montaje A y en el acuario B la distribución B. La descripción de los acuarios se encuentra en la sección 2.1.

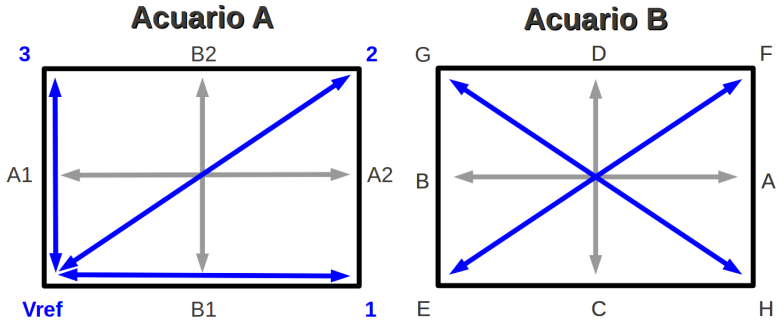

FIGURA 2.12: Dos setups distintos para la disposición de dipolos para la adquisición de la señal eléctrica del pez en el acuario. A la izquierda el setup A, del que partimos en la investigación. A la derecha el desarrollado en el GNB durante la investigación y que hemos denominado setup B.

Es necesario notar que el filtro y el calentador de la pecera introducen ruido en la señal captada por los dipolos, por tanto, es necesario desconectarlos en la realización de los experimentos. En el montaje B se añadió una conexión de la masa del circuito al agua del propio acuario, de tal modo que anulaba los efectos de la tensión común.

2.3.2. Circuito amplificador-sumador

Tras la obtención de la señal por los dipolos de la pecera, es necesario un proceso de sumado de las señales diferenciales para la construcción de una señal única. Previamente realizamos también un proceso de filtrado y amplificación con el objetivo de obtener una señal lo más limpia posible. Finalmente, se realiza un cuadrado de la señal cuya salida entrará Todo este proceso se realiza mediante un circuito analógico intermedio entre la recepción de señal de la pecera y la entrada en el sistema DAQ.

Para los primeros experimentos se hizo uso del montaje diseñado y construido por el GNB en colaboración con el *Grupo de Física Computacional e Instrumentaçao Aplicada del Instituto de Física de São Carlos, Universidad de São Paulo, São Paulo*. El esquemático de diseño del circuito puede observarse en la figura 2.13. El circuito original utilizado en los experimentos se trataba de un prototipo, construido con tecnología wire-wrapping y ampliamente utilizado en el GNB. . Coincidiendo con el desarrollo de este proyecto en el GNB se implementó una solución con tecnología PCB (Printed Circuit Board) que permitía un uso equivalente, que aumentaba la funcionalidad y versatilidad del mismo y que mejoraba en robustez, fiabilidad y modularidad al prototipo inicial. Ambas implementaciones pueden observarse en la figura 2.14.

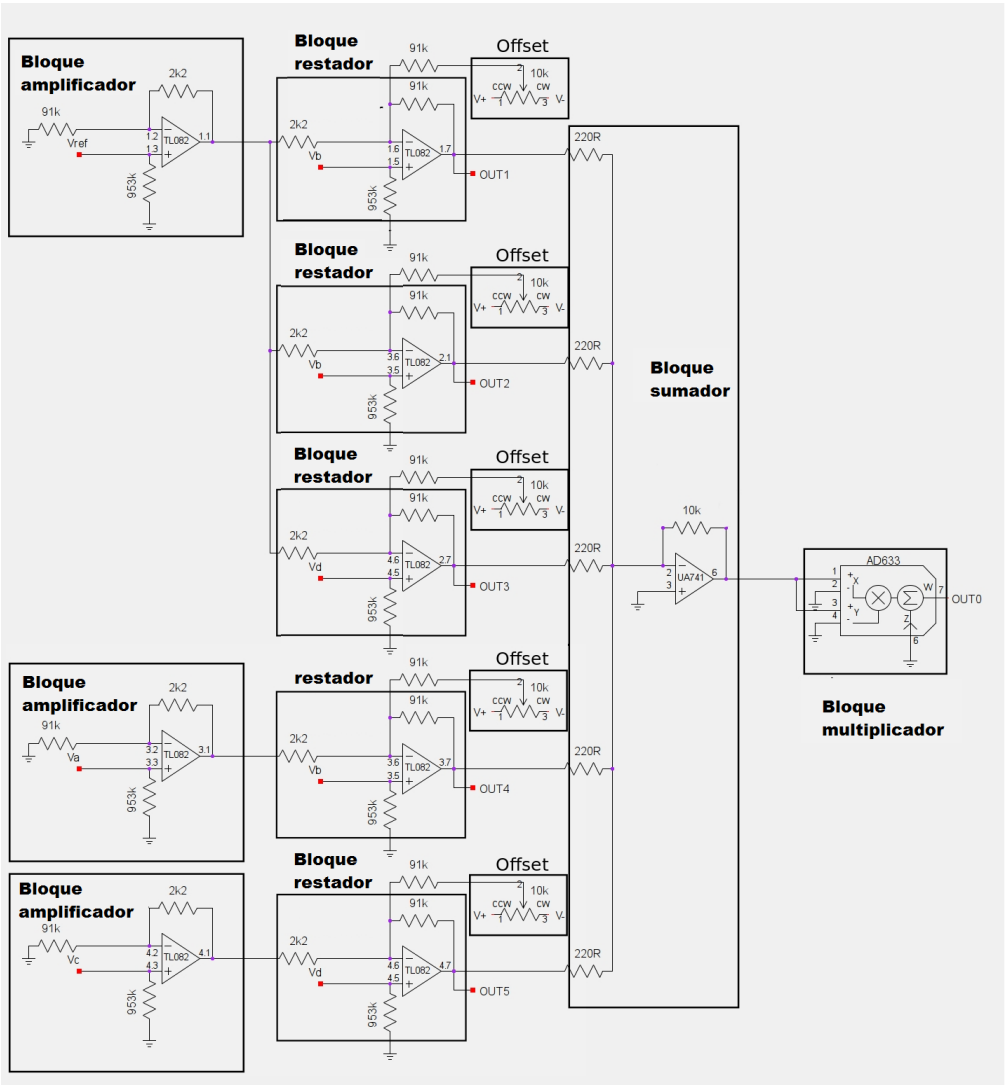

FIGURA 2.13: Esquema del circuito amplificador-sumador.

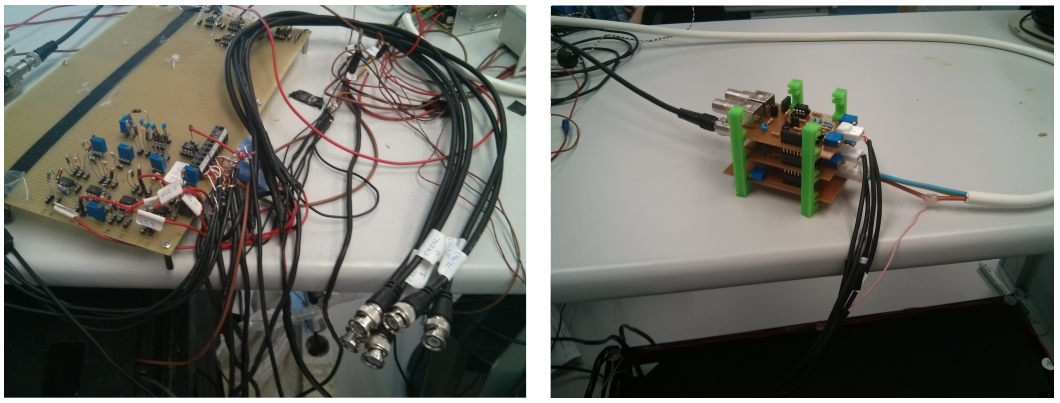

FIGURA 2.14: Dos montajes del circuito amplificador-sumador. A la izquierda el prototipo wire-wrap previo. A la derecha el montaje en PCB.

2.3.3. Tarjeta DAQ

La placa DAQ utilizada es del modelo BNC-2090A de National Instruments, que dispone de 16 canales de entrada y 2 canales de salida analógica. La salida del circuito se conecta con una de las entradas de la tarjeta. Una de las salidas analógicas de la tarjeta se conecta con el pez eléctrico artificial.

La comunicación con el PC se realiza mediante drivers de la librería COMEDI[1], tal como se indica en la seccion 2.4, que trata sobre el software utilizado. El software también permite seleccionar los canales de entrada y salida a utilizar.

2.3.4. Pez artificial estimulador

Para la emisión de estímulos en el acuario se utiliza un sistema que emula a un pez eléctrico artificial, capaz de generar campos eléctricos de un modo sencillo pero verosimil.

Se ha tomado como referencia un modelo en el cual el potencial negativo se concentra en el pedúnculo caudal mientras el resto del cuerpo del pez se encuentra cargado positivamente. Haremos uso de un dipolo emisor, con un polo situado en un extremo de una jeringa sin aguja y el otro situado al final de émbolo. De este modo podemos utilizar el movimiento del émbolo para regular la distancia entre polos del dipolo emisor, lo que modela peces de distinto tamaño. Las figuras 2.15 y 2.16 muestran el pez artificial estimulador regulado al mínimo (10cm.) y máximo (17cm.) tamaño respectivamente.

Los extremos del dipolo están hechos de plata, fijada a los cables con pegamento termofusible neutro y no tóxico.

[1] www.comedi.org

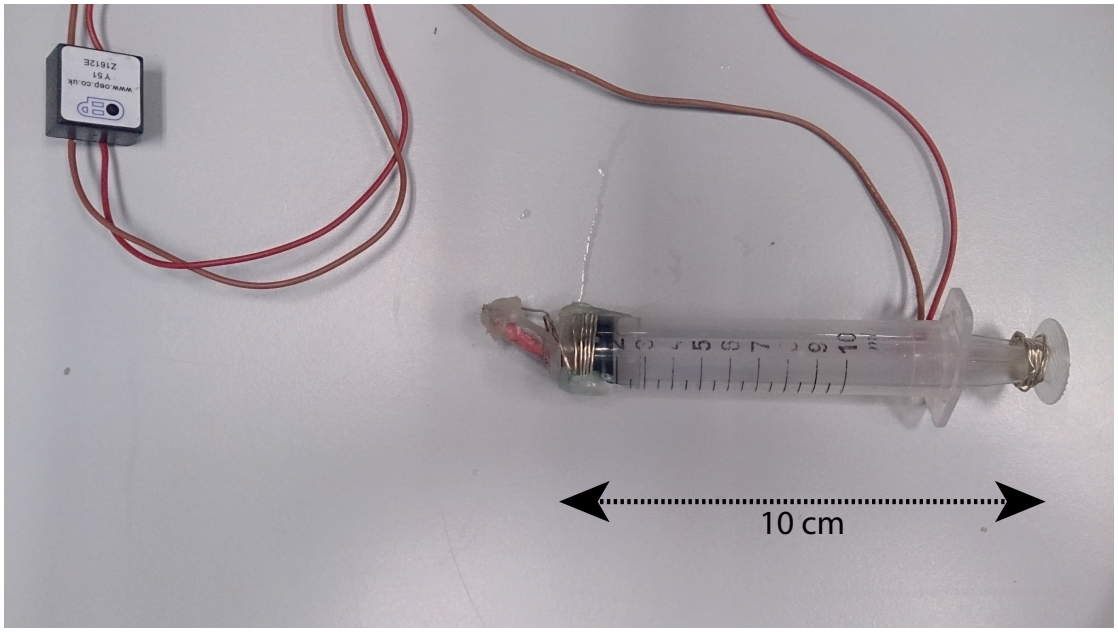

FIGURA 2.15: Fotografía del pez artificial, regulado en su amplitud mínima de 10cm, junto al correspondiente transformador de aislamiento.

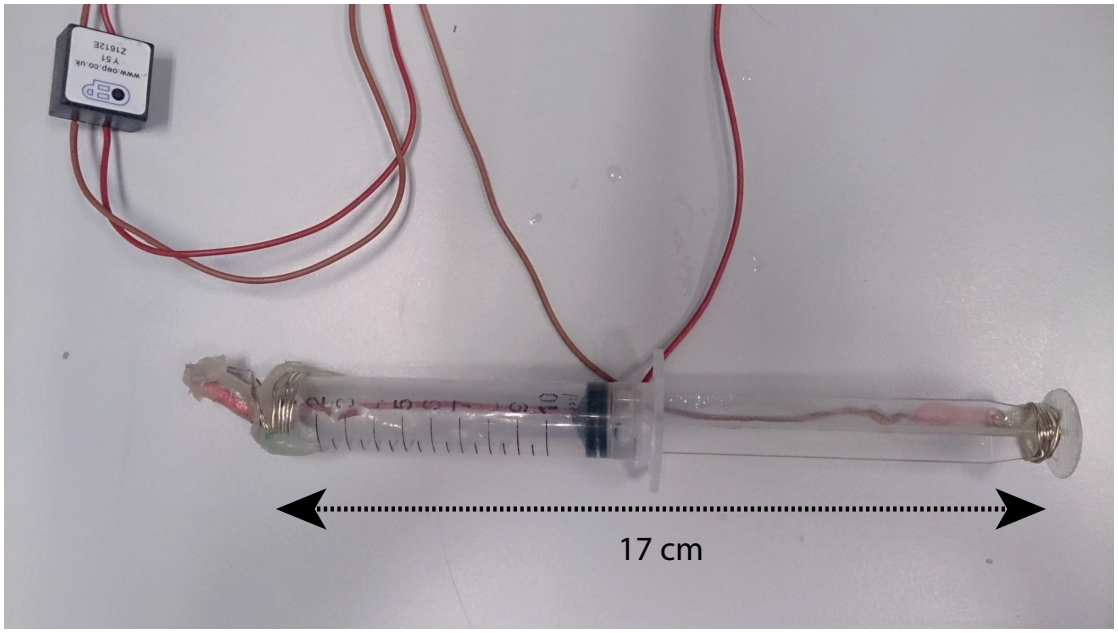

FIGURA 2.16: Fotografía del pez artificial, regulado en su amplitud máxima de 17cm, junto al correspondiente transformador de aislamiento.

La señal se hace uso de la salida analógica de la tarjeta DAQ. Para evitar derivaciones de corriente que podrían ser perjudiciales para el pez se ha intercalado un transformador de aislamiento entre la salida y el pez artificial.

2.4. Arquitectura Software

La arquitectura software utilizada en los experimentos es la encargada de permitir al usuario establecer los parámetros de los distintos experimentos a realizar, de adquirir y almacenar los datos de la señal eléctrica del pez, de procesar estos datos y de controlar la emisión de estímulos. En definitiva, puesto que controla la adquisición de datos, el procesamiento y la emisión de estímulos, es la responsable de controlar el establecimiento del ciclo cerrado.

Para que este ciclo cerrado pueda tener lugar es necesario, por un lado, permitir al usuario seleccionar los parámetros que guiarán la estimulación (lo que viene dado por los diferentes módulos del software: Estimulador, detección de palabras...). Por otro lado, de manera general es necesaria una latencia recepción-procesamiento-emisión que cumpla los requerimientos temporales establecidos para el sistema biológico que estamos estudiando (lo que conseguimos mediante el uso de la tecnología en tiempo real).

Los retrasos del sistema deben ser predecibles, luego el sistema debe operar en tiempo real. Como ya se ha comentado en la sección 1.3.1, un sistema en tiempo real se define como aquel capaz de garantizar los requerimientos temporales de aquellos procesos que están bajo su control. Esto significa que el sistema debe disponer de la capacidad de responder a eventos de manera prácticamente instantanea y temporalmente predecible. Retardos inesperados en la respuesta del sistema pueden tener un alto coste. Para el campo de aplicación de nuestro sistema, podemos definir tiempo real como la capacidad del software a utilizar de adquirir datos, analizarlos y simular comportamientos biológicos en un computador a la misma velocidad que dicho proceso tendría lugar de manera natural en la vida real.

Algunos de los proyectos software que implementan un ciclo cerrado en tiempo real como el descrito y que pueden utilizarse en la comunicación con sistemas biológicos son Extended Dynamic Clamp, RTLDC, MRCI y LabVIEW-RT Dynamic Clamp [50, 55]. En este proyecto se ha utilizado Advanced Dynamic Clamp (también conocido como ADClamp o RTBiomanager), que ha sido diseñado en el GNB como un software de propósito general para el establecimiento de ciclos cerrados en tiempo real con sistemas biológicos.

Dicho software está implementado para ejecutarse en un sistema operativo Linux con un kernel parcheado con RTAI. Real Time Application Interface (RTAI) es una extensión para el kernel de Linux que permite a este sistema operativo funcionar como un sistema operativo en tiempo real, de manera que aporta respuestas deterministas a determinadas interrupciones. Esto quiere decir que permite la ejecución de aplicaciones que cumplan de manera estricta las restricciones y los requisitos temporales. Consta de dos partes: Por un lado, una API para la programación de aplicaciones en tiempo real; por otro, un parche para el kernel de Linux que permite la ejecución de las aplicaciones implementadas con la interfaz de desarrollo. Más información sobre RTAI está disponible en el Apéndice A.

El desarrollo de SW realizado específicamente para este proyecto ha consistido, además de en mejoras concretas de los módulos ya existentes y la GUI; en la implementación del módulo words con sus diferentes subopciones: el cálculo y visionado de histogramas de palabras binarias de manera on-line, la estimulación basada en la emisión por parte del sistema biológico de determinadas palabras binarias y, finalmente, el módulo que realiza el protocolo de estimulación aleatoria. Los protocolos de estimulación ha sido explicados en la sección 2.2.3.

2.5. ADClamp

La aplicación Advanced Dynamic Clamp (ADClamp) permite la interacción bidireccional entre sistemas biológicos (neuronas,circuitos neuronales, sistemas neuronales o cerebros) y artificiales. Este software ya ha sido utilizado para aplicaciones de interacción en ciclo cerrado que hacían uso de microinyecciones, de dispositivos mecánicos o de estimulación dirigida por eventos de video [1].

Esta comunicación se basa fundamentalmente en la tecnología en tiempo real, en concreto en el concepto del observador/controlador dinámico en tiempo real (RTDOC, Real Time Dynamical Observer and Controller) que consiste en la interacción con sistemas biológicos para observarlos y controlarlos mediante un bucle de estímulo/respuesta. Como ya se ha comentado, ADClamp trabaja sobre un sistema operativo Linux parcheado con RTAI.

La tecnología de tiempo real es necesaria para que el sistema sea capaz de adaptarse a las características no lineales y variantes en el tiempo de los sistemas biológicos. La aplicación del concepto de tiempo real resulta imprescindible para superar la barrera entre los sistemas biológicos e informáticos. El sistema elegido trabaja a una frecuencia de 15 KHz, por lo que tenemos un periodo de muestreo de 66.6ns, suficiente para la

aplicación al estudio del comportamiento del pez eléctrico (ya que el tiempo medio de un pulso de la especie *Gnathonemus Petersii* es de 0.8ms.)

2.5.1. Interfaz gráfica de usuario (GUI)

ADClamp cuenta con una interfaz gráfica de usuario, conocida también como GUI (del inglés Graphical User Interface) que muestra la información del sistema y las acciones disponibles. Esta GUI posibilita, a través del uso y la representación de un conjunto de imágenes y objetos gráficos, una interacción amigable con el sistema informático.

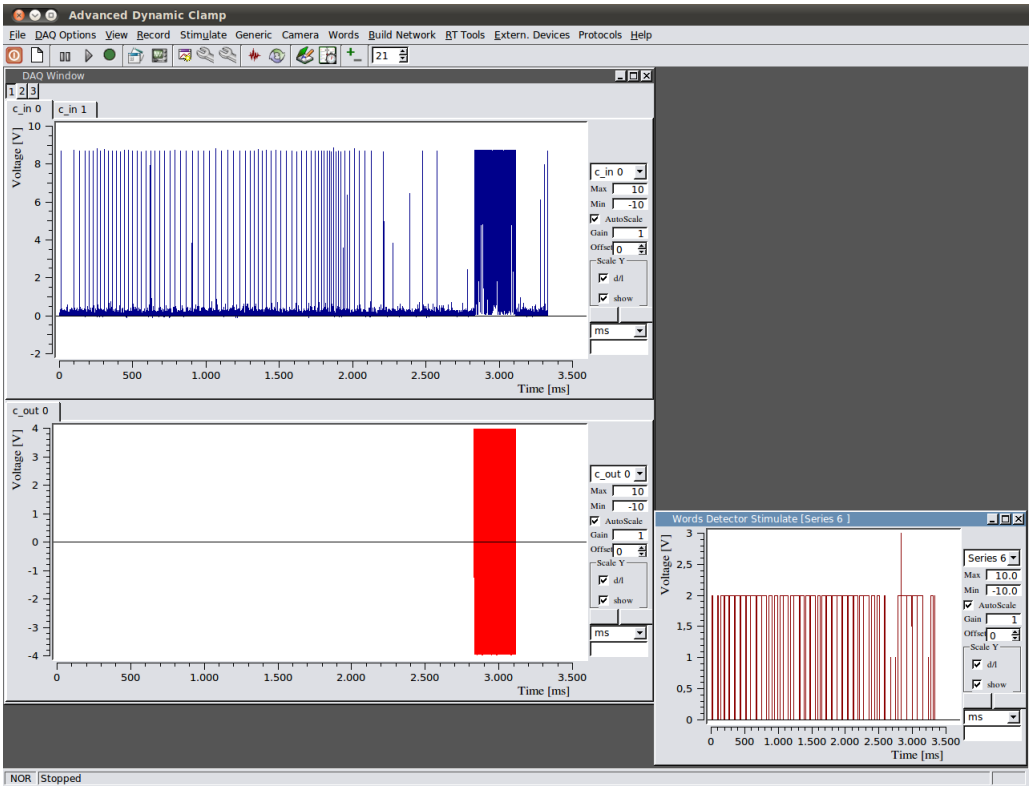

FIGURA 2.17: Ventana principal de ADClamp.

El desarrollo de la interfaz se ha realizado con Qt[2]. Qt es una biblioteca multiplataforma orientada a objetos y basada en C++, que permite el desarrollo de interfaces gráficas de usuario. Un aspecto interesante de esta tecnología, es que posee widgets, como Qwt[3].

Qwt es una extensión o widget de Qt para el desarrollo de aplicaciones científicas. En concreto, en nuestro proyecto Qwt se ha utilizado para la representación gráfica de señales, por ejemplo, aquella adquirida de la tarjeta DAQ o la enviada para estimulación. Un ejemplo de su uso se puede observar en la figura 2.17.

[2]http://www.qtsoftware.com
[3]http://qwt.sourceforge.net/

La ventana principal de ADClamp por defecto nos muestra dos gráficas, correspondiente a la adquisición de datos por un canal de entrada (c_in 0) y la emisión realizada por un canal de salida (c_out 0). Podemos también cambiar el canal que se visualiza pinchando en la pestaña c_in 1, de manera que visualizaríamos el canal 1 de entrada.

La opcion DAQ Settings nos permite configurar la tarjeta DAQ para la adquisición y la emisión de datos. Desde ahí podemos seleccionar qué canales de entrada y salida utilizar, así como la frecuencia de muestreo. Podemos acceder a ella pulsando en el botón DAQ Settings del menú de herramientas, en DAQ Options¿DAQ Settings o al utilizar el atajo de teclado Ctrl+D. Más información sobre el funcionamiento de esta ventana en la subsección 2.5.2.

En la barra de menús nos encontramos con el submenú Stimulator, que nos da acceso al módulo estimulador. Dicho módulo nos permite la construcción de los estímulos a emitir y está explicado en la sección 2.5.3.

Por último, destacamos también el módulo Words, desarrollado específicamente para este proyecto. Se accede a él a través del submenú Words de la barra de menús y cuenta con tres opciones:

- Generador de Histogramas
- Detector de palabras
- Estimulación aleatoria

Como es de esperar, implementa la binarización de la señal eléctrica del pez y su codificación a palabras binarias; también implementa un protocolos de estimulación en ciclo cerrado guiado por códigos binarios y un protocolo de estimulación aleatorio que replica al ciclo cerrado. Finalmente, también nos da la opción de observar de manera online los histogramas de palabras generados por el pez eléctrico mediante su actividad electrorreceptiva.

El funcionamiento de este módulo se encuentra convenientemente descrito en la sección 2.6.1.

2.5.2. Adquisición de datos

Como ya hemos dicho DAQ Settings nos permite configurar la tarjeta para la adquisición y la emisión de datos. El diálogo que nos presenta la interfaz puede observarse en la figura 2.18.

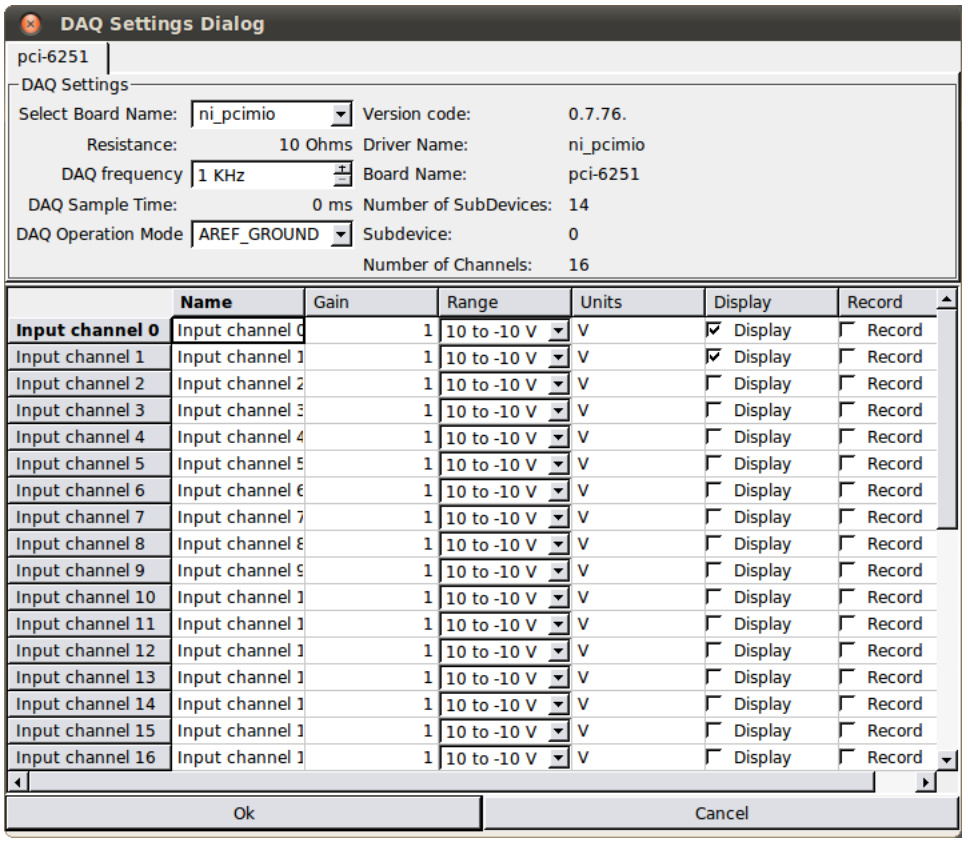

FIGURA 2.18: Diálogo de DAQ Settings.

En esta ventana podemos seleccionar la tarjeta DAQ a utilizar (Select Board Name), la frecuencia con que queremos que muestree (DAQ Frequency) y el modo de operación (DAQ Operation Mode). También podemos configurar los canales que queremos mostrar (Activar/Desactivar display), su nombre (Columna Name), rango, ganancia (Columna Gain), etc. Los modos de operación definen el valor de referencia a utilizar (tierra será el habitual, común, diferencial entre entradas u otra).

La comunicación con la tarjeta se realiza mediante los drivers de comedi [4] es un conjunto de librerías y drivers open-source para diversas tarjetas de adquisición. El manejo de esta tecnología en conjunción con RTAI permite el envío y la recepción de señales en tiempo real mediante una tarjeta DAQ.

2.5.3. Módulo estimulador

El módulo estimulador es el encargado de permitir al usuario establecer los parámetros del tipo de estimulo a presentar al sistema biológico. El diálogo de la interfaz de usuario permite al usuario la construcción de estimulos complejos, que pueden ser por ejemplo

[4] http://www.comedi.org

una concatenación de estímulos simples. El diálogo del estimulador no tiene restricciones temporales estrictas, ya que la comunicación con el usuario se realiza por el teclado y el ratón del PC, por lo que no necesita operar en tiempo real.

Una vez seleccionado el estímulo, este módulo es el encargado de generarlos y transmitirlos al canal de salida de la tarjeta para su emisión mediante el pez artificial. La generación y transmisión del estímulo sí que tiene restricciones temporales estrictas, por lo que estas tareas han de funcionar en tiempo real.

El diálogo del estimulador puede observarse en la figura 2.19

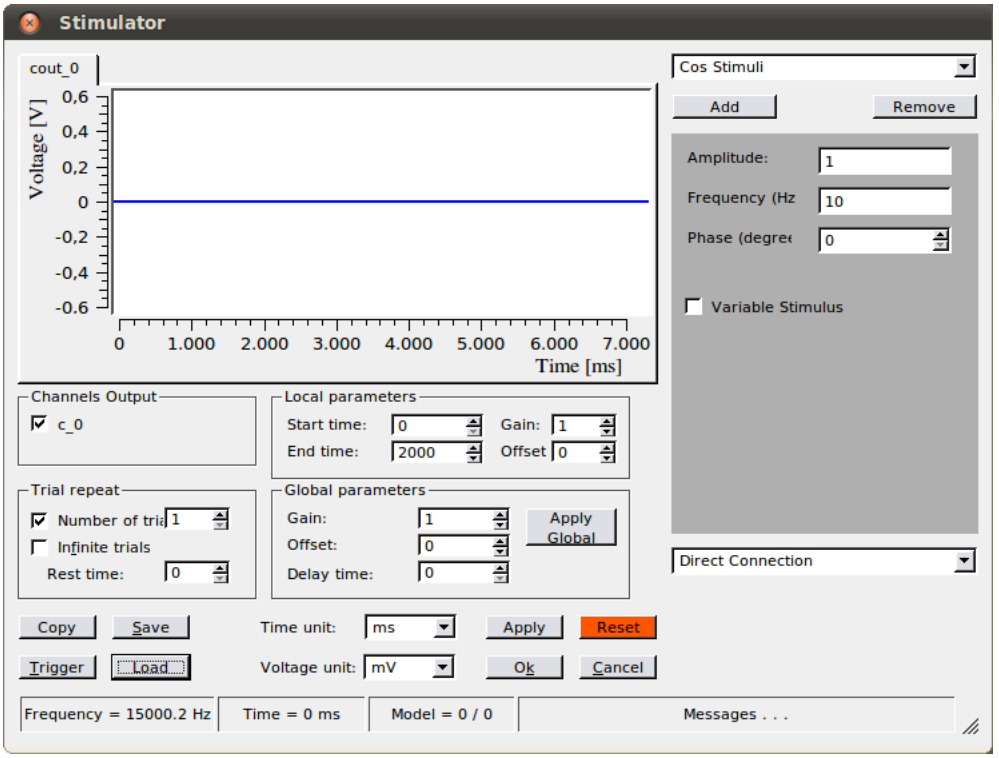

FIGURA 2.19: Diálogo de Stimulator para la construcción de estímulos en ADClamp.

Tipo de estímulo. En la zona derecha de la ventana *Stimulator* podemos seleccionar el modelo de estímulo a añadir. Cada modelo tiene sus propias opciones, de modo que si seleccionamos un modelo seno, podremos definir la amplitud, la frecuencia o el ángulo de fase en grados, tal como puede verse en la imagen 2.20. Si por ejemplo elegimos cargar un estímulo desde archivo, nos permitirá seleccionar el archivo a cargar, lo que puede verse en la imagen 2.21.

Local parameters. En esta sección definimos el tiempo de inicio y final del siguiente modelo de estímulo a incluir. También podemos definir un offset (desplazamiento respecto al eje Y) y una ganancia (multiplicador) locales.

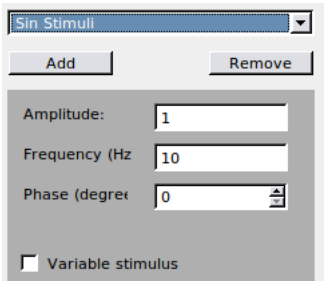

FIGURA 2.20: Subdiálogo tipo de estímulo para la construcción de estímulos tipo seno en ADClamp.

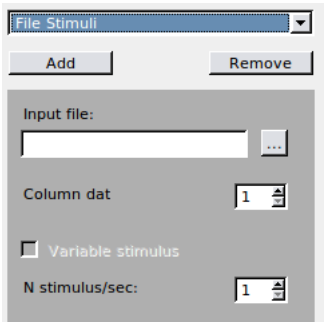

FIGURA 2.21: Subdiálogo tipo de estímulo para la construcción de estímulos cargados desde archivo en ADClamp.

Global parameters. Permite definir una ganancia, un offset y un tiempo de retardo que se apliquen globalmente. Una vez definidos, basta con pulsar en el botón Apply Global para aplicarlos. Estos valores globales se aplicarán únicamente a los modelos de estímulos que añadamos tras su aplicación, por lo que no modificarán los modelos de estímulos ya incluidos en el estímulo generado.

Channels Output. Permite marcar el canal de salida por el que deseamos que se envíe el estímulo.

Trial repeat. Numero de repeticiones del estímulo. Si marcamos infinite trials el estímulo definido se repetirá continuamente. *Rest time* establece un tiempo de espera entre repeticiones.

2.6. Desarrollo software durante el proyecto

A lo largo de este proyecto se han desarrollado diferentes herramientas en tiempo real integradas en el software ADClamp. Dicha implementación surge como resultado a la necesidad de realizar experimentos con nuevas técnicas de estimulación en ciclo cerrado

guiadas por códigos o palabras binarias. Estas herramientas se han implementado en un nuevo módulo denominado Words.

Aparte del módulo Words y de otras mejoras implementadas en ADClamp, se han desarrollado una serie de programas y scripts para la realización de los análisis necesarios para la investigación. La más importante es el programa PezHist para la generación offline de estadísticas relacionadas con la emisión de palabras binarias y la entropía asociada a las distintas distribuciones de emisión de estos códigos. Finalmente, se han implementado una serie de utilidades para realizar diferentes análisis sobre la señal y para presentar algunos de los resultados de los experimentos, dichas herramientas se describen en la subsección 2.6.3.

2.6.1. Módulo ADClamp Words

ADClamp no contaba inicialmente con ningún módulo para el establecimiento de ciclos cerrados basados en la emisión de códigos o palabras binarias. Por ello, se ha debido implementar esta funcionalidad durante el desarrollo del proyecto.

Por tanto, el módulo ADClamp Words es el encargado de controlar los experimentos basados en estimulación en ciclo cerrado guiado por códigos o palabras binarias.

A la hora de describir la funcionalidad y los requerimientos del módulo se han establecido los requisitos software que se listan en las tablas 2.2 y 2.3:

Cuadro 2.2: Requisitos NO FUNCIONALES

Código	Nombre	Descripción
OF	Offline	Sin restricciones temporales
ON	Online	Actualización constante sin restricciones temporales estrictas
TR	Tiempo real	Actualización constante y restricciones temporales estrictas.

Este módulo consta de dos partes. Por un lado, una serie de diálogos en la interfaz de usuario y por otro un módulo del núcleo, destinado a ejecutarse como tarea en tiempo real.

Los diálogos GUI permiten al usuario el establecimiento de parámetros a la aplicación. Dichos diálogos se integran en la GUI de ADClamp y han sido construidos, como el resto de la GUI, en C++ mediante Qt.

Respecto a la parte destinada a ejecutarse como tarea en tiempo real, es la que implementa la funcionalidad real del módulo. Para ello, requiere de los parámetros introducidos por interfaz de usuario. Como en el resto de módulos de ADClamp, la comunicación de

CUADRO 2.3: Requisitos FUNCIONALES

Código	Descripción	Relacionados
F1	Codificación de la señal (descripción en 2.2.2)	TR
F1.1	Selección de tiempo de bin	OF
F1.2	Selección de tamaño de palabra	OF
F2	Generación de histogramas de distribución de códigos	TR, F1
F2.1	Selección de tiempo de bin	OF
F2.2	Selección de tamaño de palabra	OF
F2.3	Histograma del periodo instantaneo	ON, F1
F2.3.1	Selección de parámetro tiempo instantaneo Ti	OF, T1
F2.3.2	Actualización	ON
F2.4	Histograma acumulado	ON, T1
F2.4.1	Actualización	ON
F2.5	Impresión de histogramas por pantalla	ON
F3	Detección de códigos y lanzamiento de estímulo	TR
F3.1	Selección de tiempo de bin	OF
F3.2	Selección de tamaño de palabra	OF
F3.3	Selección de palabra	OF
F3.4	Detección de palabra	TR, T1
F3.4.1	Actualizacion de palabra detectada	TR, T1
F3.5	Selección de retraso de estimulación	OF
F4	Estimulación aleatoria	TR
F4.1	Selección de tiempo de bin	OF
F4.2	Selección de tamaño de palabra	OF
F4.3	Selección de periodo de estimulación aleatoria	OF
F4.3.2	Actualizacion de palabra detectada	TR, T1
F4.4	Selección de retraso de estimulación	OF

los diálogos (que operan en el sistema operativo) con el módulo del núcleo (la tarea en tiempo real) se realiza mediante una memoria compartida.

Como se explica en la sección de ADClamp 2.5 y en el apéndice A, RTAI trata al núcleo standar de Linux como una tarea de menor prioridad, por lo que se ejecuta únicamente cuando no hay ninguna tarea tiempo real (de mayor prioridad) ejecutándose. Por ello, es necesario tener en cuenta el tiempo de ejecución de la tarea en tiempo real, ya que podría llegar a bloquear por completo el sistema si su duración es mayor que el tiempo asignado a cada interrupción. En nuestro caso, ADClamp trabaja a una frecuencia de 15KHz.

Con el objetivo de operar con palabras binarias se ha realizado una librería de funciones destinadas a la binarización de una señal analógica como la del pez eléctrico (detección de pulsos) y la detección de palabras binarias (almacenamiento de bits, comparación de palabras...). Dicha librería se describe en profundidad en el apéndice B.

Respecto a las funcionalidades implementadas en el módulo words, pueden dividirse en tres:

- Generador de histogramas
- Detector de palabras
- Estimulación aleatoria

El diálogo GUI del generador de histogramas puede verse en la figura 2.22. Su objetivo es construir de manera online histogramas de códigos emitidos, para ello depende de:

- Umbral de voltaje: A partir del cual se detecta un pulso. Necesario para binarizar la señal.
- Tiempo de bin: Periodo en el cual se detecta o no la existencia de un pulso. Necesario para binarizar la señal.
- Periodo: Tiempo de actualización de los histogramas que se muestran por pantalla.

Hace uso de la librería de palabras para la binarización de la señal y la generación de palabras. Mediante una tabla muestra por pantalla un histograma instantaneo de las palabras emitidas en el último periodo; así como un histograma acumulado del número de palabras desde el comienzo de la ejecución. Puede verse un ejemplo en la figura 2.23. La frecuencia de actualización viene dada por el periodo introducido en la interfaz de usuario. La tarea en tiempo real almacena únicamente los datos de palabras que se generan en un determinado periodo y, terminado este, envia esos datos mediante memoria compartida al programa ejecutándose en espacio de usuario. Este se encarga de tomar esos datos y actualizar los histogramas que se muestran por pantalla.

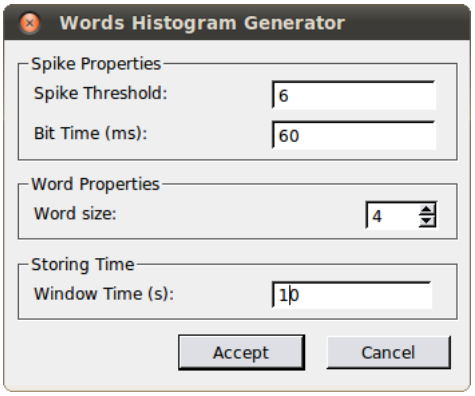

FIGURA 2.22: Diálogo GUI del generador de histogramas (Módulo Words de AD-Clamp).

Histogram Table

	Instant Word	Instant #	Accum. Word	Accum. #
1	10	29	10	249
2	5	24	5	221
3	4	16	15	207
4	13	14	4	178
5	6	14	8	153
6	9	12	1	153
7	2	11	2	147
8	11	9	6	138
9	3	8	0	137
10	8	6	13	131
11	1	6	11	97
12	0	4	3	89
13	14	3	9	77
14	12	3	12	54
15	7	3	14	45
16	15	1	7	43
17				
18				
19				
20				

FIGURA 2.23: Diálogo GUI anexo del generador de histogramas con tabla de histogramas (Módulo Words de ADClamp).

Respecto al detector de palabras, su funcionalidad consiste en detectar que se ha generado (por parte del sistema biológico bajo estudio) una palabra binaria determinada y lanzar el estímulo que se encuentre cargado en el estimulador. El diálogo para la introducción de los parámetros del detector de palabras puede observarse en la figura 2.24. Requiere de la introducción de los siguientes parámetros:

- Umbral de voltaje: A partir del cual se detecta un pulso. Necesario para binarizar la señal.
- Tiempo de bin: Periodo en el cual se detecta o no la existencia de un pulso. Necesario para binarizar la señal.
- Periodo: Tiempo de actualización de los histogramas que se muestran por pantalla.

Finalmente, los parámetros de la estimulación aleatoria se controlan mediante un diálogo como el que puede verse en la figura 2.25.

2.6.2. Generación de histogramas offline: PezHist

Para el análisis offline de la señal eléctrica del pez se ha desarrollado un programa denominado PezHist. Este programa toma como entrada un fichero generado por ADClamp en la adquisición de señal y genera, a partir de los parámetros que se le suministran,

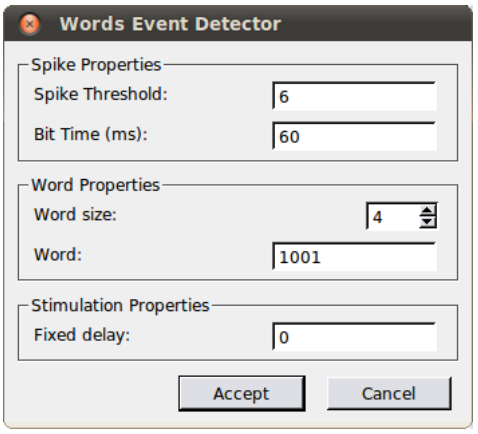

FIGURA 2.24: Diálogo GUI anexo del detector de palabras (Módulo Words de AD-Clamp).

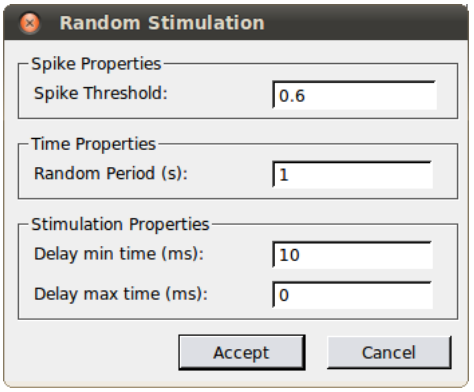

FIGURA 2.25: Diálogo GUIdel estimulador aleatorio (Módulo Words de ADClamp).

un fichero de estadísticas con las palabras emitidas, el número de ocurrencias de cada palabra y la entropía de la distribución de palabras.

PezHist permite generar varios histogramas (y calcular las entropías asociadas a sus distribuciones) al mismo tiempo en función de distintos valores de los parámetros Δt y wl. La salida de parámetros se realiza en un fichero de estadísticas HDF5[5]. Esta tecnología permite un manejo simple de colecciones grandes de datos.

La entrada de parámetros se realiza mediante un fichero de configuración. El fichero de configuración se encuentra escrito en YAML , que es un formato de serialización de datos legible para humanos.

El código de la aplicación PezHist, asi como un ejemplo de fichero de configuración YAML puede consultarse en el apéndice D

[5]http://www.hdfgroup.org/HDF5/

2.6.3. Utilidades

Para la ejecución de los análisis listados en este proyecto se han realizado una serie de aplicaciones, habitualmente en forma de scripts. Estas utilidades realizan operaciones de bajo calado pero esenciales para el desarrollo de los experimentos, la generación de resultados (y gráficas que los muestren) y la obtención de conclusiones.

Se realiza aquí un pequeño listado de las utilidades realizadas que pueden tener algún interés, si bien se han utilizado algunas más para la realización de gráficas y figuras, asi como para la automatización de procesos de análisis. Puede consultarse el código de las utilidades mencionadas en el apéndice C.

SpikeTimes: Aplicación en C++ encargado de detectar pulsos en un fichero de señal y de devolver los tiempos en que dichos pulsos tienen lugar. Recibe como parámetro de entrada un fichero de adquisición de señal generado con ADClamp y genera un fichero de salida con los tiempos en que se ha detectado pulso.

ImpulseTimes: Aplicación C++ encargado de detectar impulsos de estimulación en un fichero de señal y de devolver los tiempos en que dichos impulsos son emitidos. Recibe como parámetro de entrada un fichero de adquisición de señal generado con ADClamp y genera un fichero de salida con los tiempos en que se ha detectado pulso.

PreImpulseSpikeDistribution: Script para Octave encargado de calcular la distribución de retrasos pulso-estímulo. Recibe un fichero de tiempos de pulso y un fichero de tiempos de impulsos de estimulación y, con ellos, calcula un histograma de retrasos. A partir de dicho cálculo genera y devuelve un gráfico de barras que muestra la distribución de los retrasos.

Capítulo 3

Experimentos y resultados

En esta sección se describen los experimentos realizados durante el desarrollo del proyecto y se muestran y discuten sus resultados.

Todos los experimentos, excepto los destinados al estudio de la evolución entrópica (Sección 3.1), están divididos en partes diferenciadas. En dichas partes obtenemos la señal emitida por el pez bajo condiciones de estimulación distintas, en función de aquello que deseamos estudiar. Normalmente cada una de estas partes tiene una duración equivalente al resto, lo cual no quiere decir necesariamente que el tiempo de cada parte sea exactamente el mismo, si no que se define la equivalencia en función de cada experimento.

Hemos considerado como condiciones normales o de control aquellas partes donde los peces se encuentran nadando libremente y donde no reciben estímulos artificiales por parte del estimulador. Las condiciones de estimulación varían en cada tipo de experimento, en función de lo que deseamos estudiar, por lo que están explicadas en la sección correspondiente a cada uno.

Es esencial para una correcta comprensión de los experimentos realizados entender la transformación de la señal analógica del pez a códigos binarios que se explica en 2.2.2.

3.1. Evolución de la distribución de códigos emitidos y de la entropía en sistemas sin estimulación

3.1.1. Descripción del experimento

El objetivo de estos experimentos es determinar cuál es el comportamiento del pez a nivel de emisión de palabras o códigos binarios en condiciones normales sin estimulación.

Estos experimentos nos sirven, por un lado, como estudio preliminar del comportamiento eléctrico de cada especimen; por otro lado, ese estudio nos permite seleccionar unos parámetros adecuados de tiempo de bin y de tamaño de palabra binaria. Una vez determinados dichos parámetros, podemos estudiar la variación del estado eléctrico a lo largo del tiempo.

Por tanto, el experimento consiste en el registro de la señal eléctrica del pez durante periodos largos de tiempo en condiciones normales, esto es, el pez en un acuario individual nadando libremente en el que no se emiten estímulos artificiales. Los datos almacenados se dividen en ventanas de tiempo de 30 minutos para evaluar la evolución temporal.

La caracterización del comportamiento la vamos a realizar en función de los siguientes indicadores de patrones de comportamiento:

- Distribución de códigos emitidos en función de los parámetros tiempo de bin y tamaño de palabra.
- Evolución de la entropía de dicha distribución en cada ventana temporal a lo largo de todo el periodo:
  - Entropía máxima.
  - Entropía media.
  - Varianza de la entropía.
  - Tiempo de bin que maximiza la entropía

La señal adquirida es procesada de manera off-line para obtener la caracterización en base a los indicadores descritos. El método para el cálculo de la entropía ya se ha especificado en la sección 2.2.2. Se realiza mediante el programa descrito en 2.6.2.

3.1.2. Experimentos realizados

El identificador de experimentos sigue la siguiente codificación:

[numero de experimento]_[acuario A/B]

- **Número de experimento:** Por orden temporal, un número único que identifica al experimento
- **Control diurno/nocturno:** Periodo del día en que tuvo lugar la grabación.
- **acuario A/B:** Acuario seleccionado para la adquisición de señal. Cabe recordar que a cada acuario le corresponde un esquema de montaje de los dipolos, tal como se explicó en la sección 2.3.1.

Ls experimentos realizados se han listado en el cuadro siguiente:

Código	Figura asociada	Comentario
1_A	3.1	
2_B	3.2	
3_B	3.5	

CUADRO 3.1: Listado de experimentos del tipo 3.1: Evolución de la distribución de códigos emitidos y de la entropía en sistemas sin estimulación.

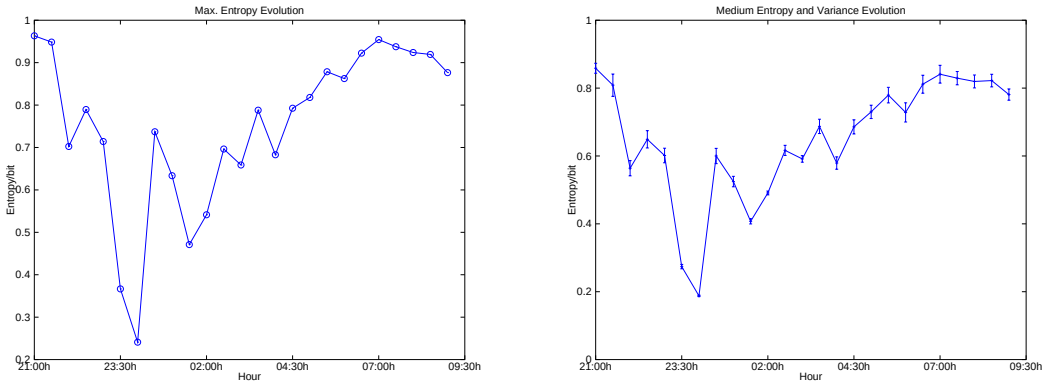

FIGURA 3.1: Evolución de los parámetros de la entropía en el experimento 1_A.

3.1.3. Discusión de resultados

Los gráficas analizan la evolución de la entropía por bit en el sistema, en base a los histogramas de palabras calculados sobre la señal adquirida para un tiempo de palabra de 4 bits y un rango de Δt desde 20ms a 210ms, en intervalos de 10ms. Se ha podido comprobar que, en general, la variabilidad de esta entropía por bit para palabras de

distinto número de bits (wl) da lugar a una evolución temporal muy similar, tanto de la entropía máxima como de la entropía media. Esto puede observarse en la figura 3.4.

Podemos partir del estudio de las gráficas donde se observa la evolución de la entropía máxima. Sobre las mismas se puede concluir que la entropía tiende a mantenerse alta, en valores por encima de 0.9, pero que sin embargo cae en determinados instantes a valores bajos. Esto puede observarse en la figura correspondiente al experimento 1_A (figura 3.1). Esta caida puede estar justificada por una actividad repetida del pez en lo que corresponde a la emisión de palabras (o, de manera subyacente, a la temporalidad en la emisión de pulsos). En este caso concreto se corresponde con una baja actividad eléctrica del pez, como puede observarse en los histogramas mostrado a continuación.

Otro aspecto a considerar sería el tiempo de bin asociado a la entropía máxima y cómo este varía a lo largo del tiempo.Aunque existe cierta estabilidad respecto al bin que maximiza la entropía, con frecuencia se presentan cambios temporales. Puede notarse facilmente que el tiempo de bin asociado a la entropía máxima varía con el tiempo. Esto, de seguir estrictamente un criterio de entropía máxima nos llevaría a modificar repetidamente el parámetro Δt respecto a los códigos con los que estimular en los experimentos con estimulación guiada por códigos. En conclusión, no está de más un análisis de la superficie entrópica del sistema previo a cada experimento como caracterización del estado eléctrico del pez. Este control puede servirnos para determinar si el sistema biológico se encuentra en unas condiciones fuera de lo habitual, lo que se manifestaría como valores de entropía anormalmente bajos o tiempos de bin que maximizan la entropía desplazados respecto a lo habitual. Es por ello que se realizarán en los experimentos subsiguientes una parte de control previa a cada experimento y, a partir de los resultados de la superficie entrópica de estos controles, se establecerán los parámetros de tiempo de bin y tamaño de palabra.

Cabe notar que el criterio de máxima entropía no determina más que una capacidad grande en el canal para transmitir información, pero no que dicha información se transmita efectivamente. Por ello, no hay necesidad de seguirlo de manera estricta y podemos seleccionar valores de entropía altos sin ser los máximos, especialmente porque la superficie entrópica no presenta pulsos, si no que suele ser bastante gradual. Podemos escoger una entropía grande que no sea necesariamente la máxima.

Si observamos la evolución del valor de la entropía asociada a un tiempo de bin que maximiza la entropía en un momento determinado, si bien no es siempre máxima, sí que se mantiene alta en todo momento. Esto se ha hecho en los experimentos subsiguientes para tratar de establecer una base comparativa entre los mismos. De tal modo, solo se cambia el valor del parámetro Δt cuando esto resulta necesario por una variación importante en la superficie entrópica. Se considera una variación importante en los parámetros

que maximizan la entropía que la selección de dichos parámetros supònga un valor de entropía por debajo del 80 % de entropía máxima.

Como trabajo futuro, un estudio continuado sobre la distribución de palabras emitidas por el pez (y la entropía resultante) durante su periodo vital podría darnos información sobre cómo evoluciona esta en el largo plazo y permitir caracterizar diferentes estados en el largo plazo.

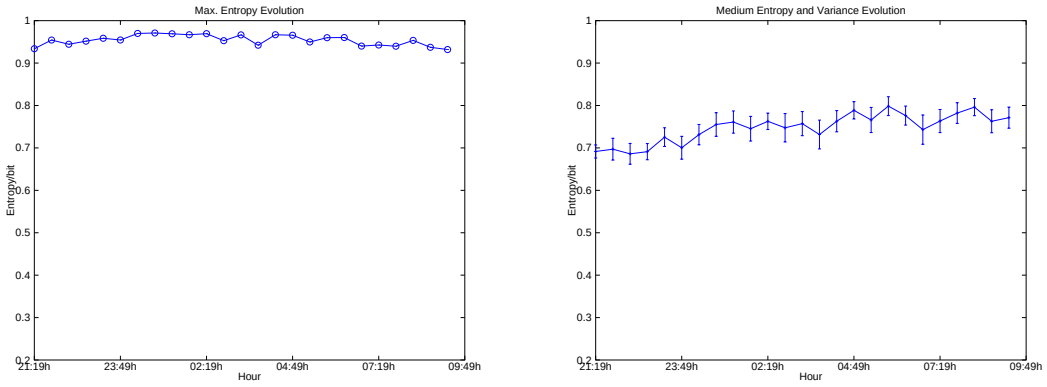

FIGURA 3.2: Evolución de los parámetros de la entropía en el experimento 2_B.

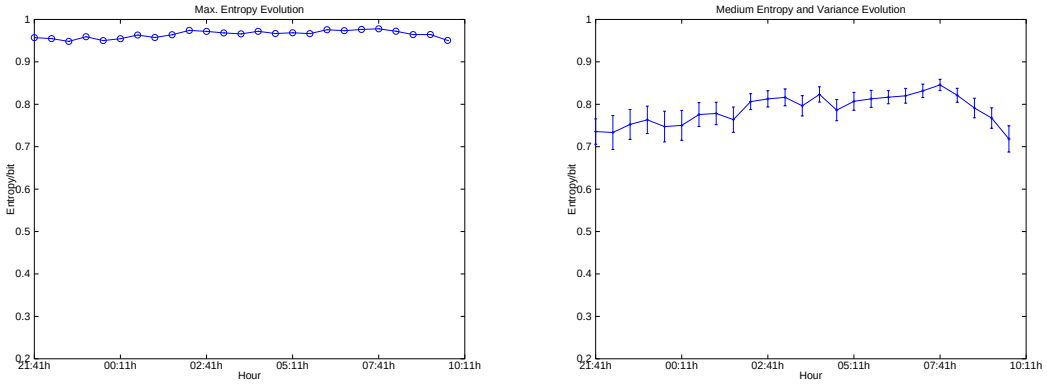

FIGURA 3.3: Evolución de los parámetros de la entropía en el experimento 3_B.

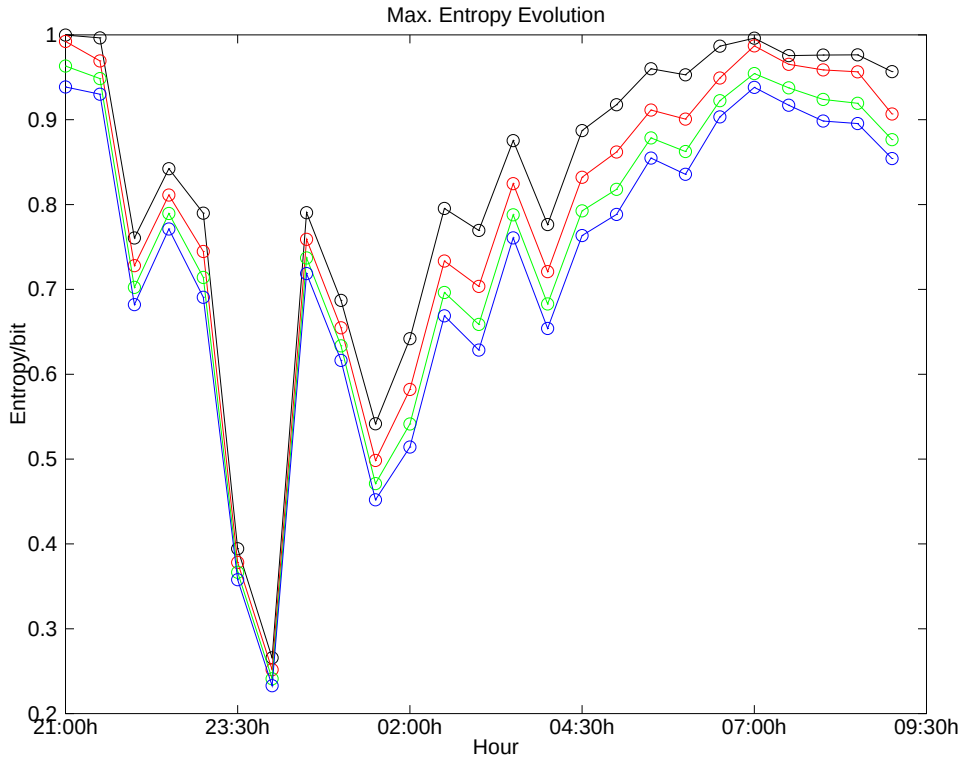

FIGURA 3.4: Evolución de los parámetros de la entropía en el experimento 1_A, diferentes tamaños de palabra.

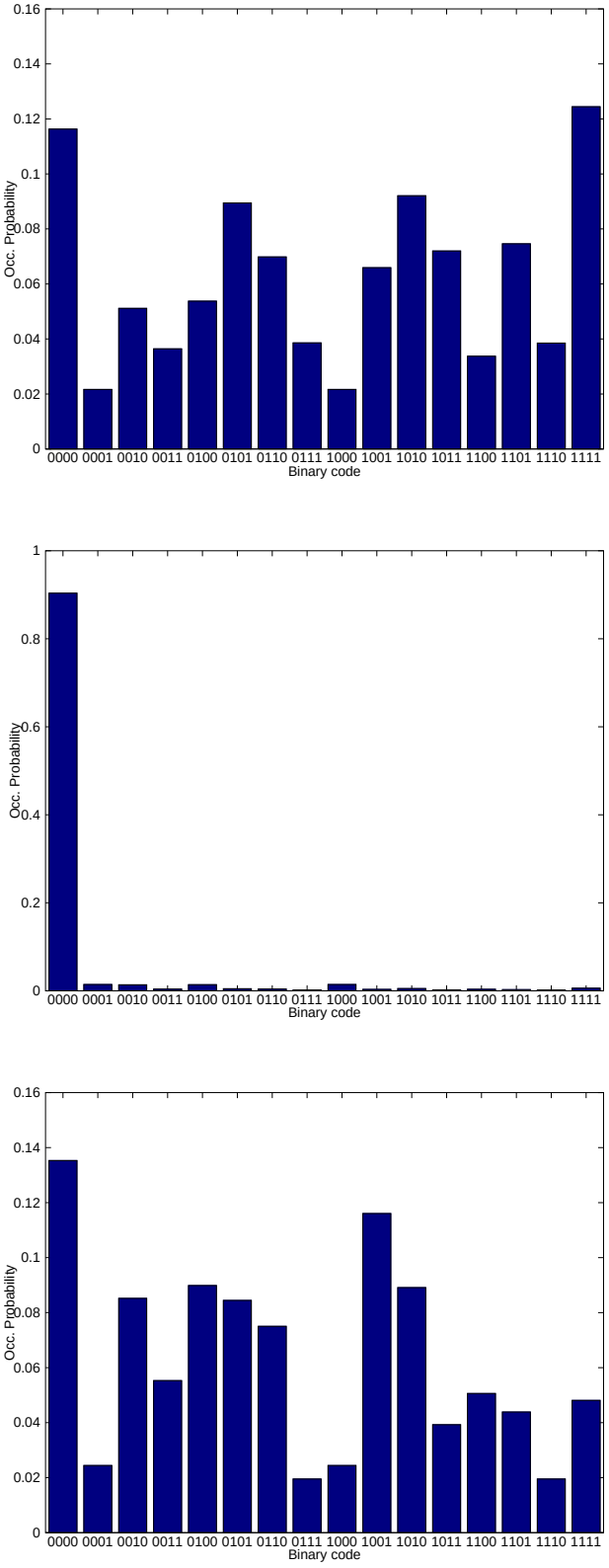

FIGURA 3.5: Histogramas de palabras en el experimento 1_B. El histograma superior corresponde a la primera media hora; el segundo al periodo que se inicia en la 3a hora de grabación, que se corresponde con el sexto punto en las gráficas de la evolución entrópica del experimento, donde la entropía se hace mínima; el último histograma corresponde al vigesimo punto en las gráficas de la evolución a largo plazo.

3.2. Análisis preliminar de la estimulación en ciclo cerrado basada en códigos (Close-loop code-driven stimulation)

3.2.1. Descripción del experimento

El objetivo de estos experimentos consiste en estudiar la respuesta del pez a una estimulación basada en códigos binarios. La definición de código o palabra binaria se encuentra en 2.2.2, así como la implementación del procedimiento de detección se encuentra descrito en 2.6.1.

Estos códigos, por cómo están construidos, portan información sobre la emisión de pulsos con una cierta temporalidad, de tal modo que toman en cuenta la posibilidad de que exista una memoria a corto plazo en la emisión de pulsos eléctricos por parte del pez y, por tanto, en los procesos de electrorrecepción. La temporalidad tomada en cuenta depende tanto del tiempo de bin como del tamaño de palabra en bits. La tesis que guía este experimento es la posibilidad de que una estimulación dependiente de la emisión de códigos concretos muestre un cambio apreciable y reproducible en el procesamiento de información eléctrica en el pez.

La existencia de cambios en el procesamiento de información bajo la presencia de estímulos artificiales ha sido mostrada en estudios preliminares realizados en este proyecto y está reflejada en algunos artículos publicados [12, 47].

Nuestro objetivo en este caso es determinar que dicho cambio depende exclusivamente de los parámetros que se pretenden estudiar (la respuesta a códigos y la existencia de una memoria a corto plazo asociada). Para ello se trata de descartar que la respuesta observada dependa, primero, de la mera presencia de estímulos artificiales y, segundo, que se trate de una respuesta a eventos instantaneos. Con este objetivo en mente se ha desarrollado un protocolo para la realización de experimentos con diferentes sesiones de control, tanto en condiciones normales como en condiciones de estimulación aleatoria.

El método de codificación de la señal analógica está explicado en la sección 2.2.2. Es de recordar que las palabras se detectan de manera superpuesta, con un desplazamiento de un solo bit. El protocolo de estimulación es el siguiente: Se inicia una ventana de binarización (de duración Δt). Durante dicha ventana se espera la presencia de un pulso. Si se detecta, se establece el bit a 1; Si el tiempo de bin se cumple sin detectar pulso, dicho bit se establece a 0. Una vez transcurrido Δt, se comprueba si la palabra activa coincide con la palabra que dispara la estimulación y, de ser así, se estimula. En cualquier caso, se reinicia el bit a 0 y se inicia una nueva ventana de binarización.

Puede comprobarse que este protocolo de estimulación implica un retraso variable entre el pulso emitido por el pez y el estímulo artificial. Dicho retraso depende de la palabra escogida. Si la palabra termina en 1, el rango del retraso estará entre $[0, \Delta t]$. En caso de que la palabra tenga n ceros al final de la misma, este rango será $[n * \Delta t, (n + 1) * \Delta t]$.

La selección de un protocolo de estimulación aleatorio se ha guiado por la necesidad de realizar una estimulación que simulase adecuadamente la que tiene lugar en la estimulación dirigida por códigos. De tal modo se decidió mantener dos parámetros:

- Ratio de estímulos por segundo
- Rango de retrasos pulso-estímulo
- Número total de estímulos

Para ello, se establecen ventanas de duración igual al periodo equivalente a la frecuencia de estímulo en la estimulación guiada por palabras. De manera que si tenemos 900 estímulos en 15 minutos de estimulación dirigida por códigos, establecemos un periodo de 1s. Transcurrido un tiempo aleatorio en esa ventana, esperamos la presencia de un pulso y, una vez detectado, esperamos un tiempo aleatorio en el rango $[0, \Delta t]$ (siendo Δt igual al utilizado en la estimulación basada en palabras). Para obtener un número total de estímulos equivalente, establecemos la duración de esta parte del experimento en tanto tiempo como sea necesario.

En la estimulación guiada por códigos se ha decidido utilizar las palabras con mayor probabilidad de ocurrencia terminadas en 1. La probabilidad de ocurrencia de una palabra viene dada por los histogramas de la distribución de códigos en el control previo de cada experimento. Existe una razón para tomar estas palabras terminadas en 1 y está relacionada con el protocolo de estimulación aleatorio. Por un lado, la existencia de ceros dificulta replicar el rango de retrasos pulso-estímulo en la estimulación aleatoria. Por otro, al establecer retrasos mayores estamos definiendo más la palabra que va a guiar la estimulación aleatoria.

Por ejemplo, si utilizamos palabras de 4 bits y establecemos un retraso pulso-estímulo en el rango $[0, \Delta t]$, la estimulación se presentará aleatoriamente tras 2^3 palabras posibles (cualquier palabra de 4 bits terminada en 1). Sin embargo, si establecemos el retraso pulso-estímulo en el rango $[3 \times \Delta t, 4 \times \Delta t]$, estaremos estimulando únicamente tras la presencia de las palabras binarias '1000' o '0000', por lo que la estimulación reduce su aleatoriedad. En el caso extremo de esperar un rango $[4 \times \Delta t, 5 \times \Delta t]$ la estimulación se presetará únicamente tras la palabra binaria de 4 bits '0000' por lo que la estimulación aleatoria resultará equivalente a la guiada por dicha palabra.

La caracterización del comportamiento la vamos a realizar en función de los siguientes indicadores de patrones de comportamiento:

- Distribución de IPIs
- Distribución de códigos emitidos
- Entropía de la distribución de códigos

3.2.2. Experimentos realizados

El identificador de experimentos sigue la siguiente codificación:

[número de experimento]_[tipo de estímulo]_[tiempo de bin]_[palabra a detectar]

- **Número de experimento:** Por orden temporal, un número único que identifica al experimento
- **Tipo de estímulo:** Forma del estímulo. Se utilizaron dos tipos, seno (sin) y pulso (pulse).
- **Tiempo de bin:** Medido en ms.
- **Palabra a detectar:** Palabra que guía el lanzamiento de un estímulo eléctrico artificial

Código	Figura asociada	Comentario
1_sin_80_0101	3.6	
2_sin_80_0101	3.7	
3_sin_80_0101	3.8	
4_sin_80_0101	3.9	
5_sin_110_1001	3.10	
6_sin_110_1001	3.11	
7_sin_110_1001	3.12	

CUADRO 3.2: Listado de experimentos del tipo 3.2: Análisis preliminar de la estimulación en ciclo cerrado basada en códigos (Closed-loop code-driven stimulation).

3.2.3. Discusión de resultados

Para el especimen 1 se decidió un tiempo de bin de 80 ms dado que era el que maximizaba la entropía en los controles previos a los experimentos. Para el especimen 2 en cambio utilizamos un tiempo de bin de 110ms, por el mismo motivo. En ambos casos se decidió un tamaño de palabra de 4 bits dado que aunque la entropía no era máxima, si era lo suficientemente grande y garantizaba un número de estados distintos suficientes. Una gráfica de la superficie entrópica resultante de un control previo a los experimentos puede verse en la figura 3.13

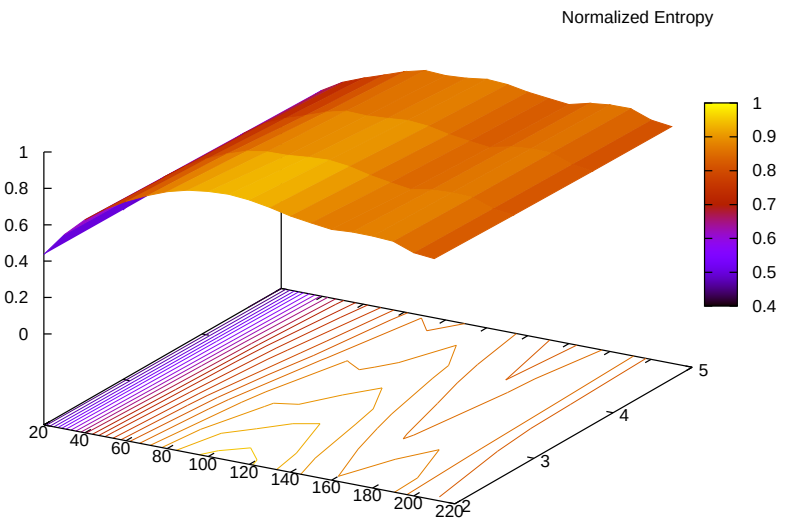

FIGURA 3.13: Entropía de la distribución de códigos emitidos durante condiciones de control, esto es, sin presencia de estimulación artificial.

Atendiendo a los resultados obtenidos podemos argumentar que existe un código de estimulación resultados obtenidos fue presentado en la conferencia

Uno de los problemas a considerar en el modelo de estimulación utilizado ha sido permitir un retraso variable entre la presencia del pulso del pez y la emisión de estímulo. Como ya se ha dicho, este sistema biológico es muy sensible a este retraso entre su pulso y una respuesta eléctrica. Un histograma de la distribución de estos retrasos tanto en el caso de estimulación basada en códigos como en el caso de estimulación aleatoria puede verse en la figura 3.14. Como puede verse, emitimos los estímulos con un retraso variable en un rango de tiempos.

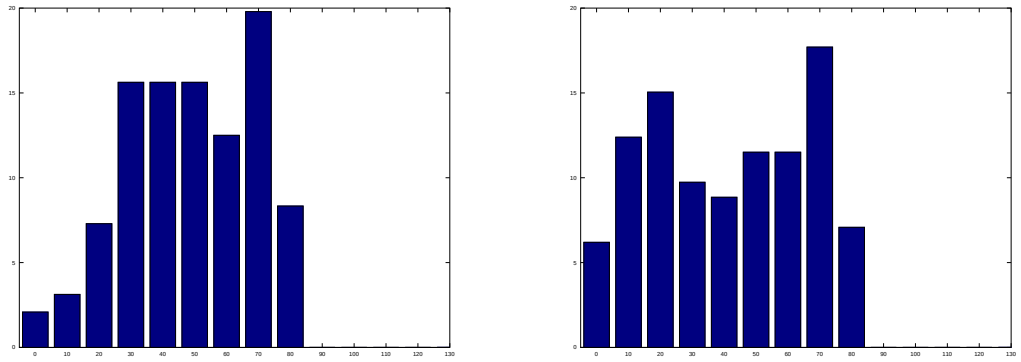

FIGURA 3.14: Ejemplo de distribución de retrasos pulso-estímulo en estimulación guiada por palabras y estimulación aleatoria. En concreto corresponde al experimento 4_sin_80_0101, de ahí que el máximo del rango sea equivalente al tamaño de bin escogido en este caso para los códigos que guían la estimulación (80ms).

Según los analizado en [47], la respuesta a la estimulación es mayor cuando el pulso se presenta con un retraso de 10ms. Por ello, se decidió realizar el análisis de la sección 3.3. En esos experimentos analizamos la respuesta, entendida como cambios en el procesamiento de información, presentando el mismo estímulo ante la detección del mismo código, pero con un retraso pulso-estímulo variable frente a un retraso fijo de 10ms.

Otro de los problemas relacionado con esta aproximación preliminar es que la comparación de resultados respecto de los cambios en el procesamiento de información derivados de una estimulación guiada por códigos de la misma longitud, aunque pueden aportar una idea preliminar de la importancia de la memoria en el sistema biológico, dificilmente van a poder aislar (y, por tanto, caracterizar) la función de la memoria. Por ello, se decidió realizar un análisis como el de la sección 3.4, que comparase la respuesta ante eventos que toman en cuenta la memoria (códigos binarios) con respecto a eventos instantaneos (pulsos).

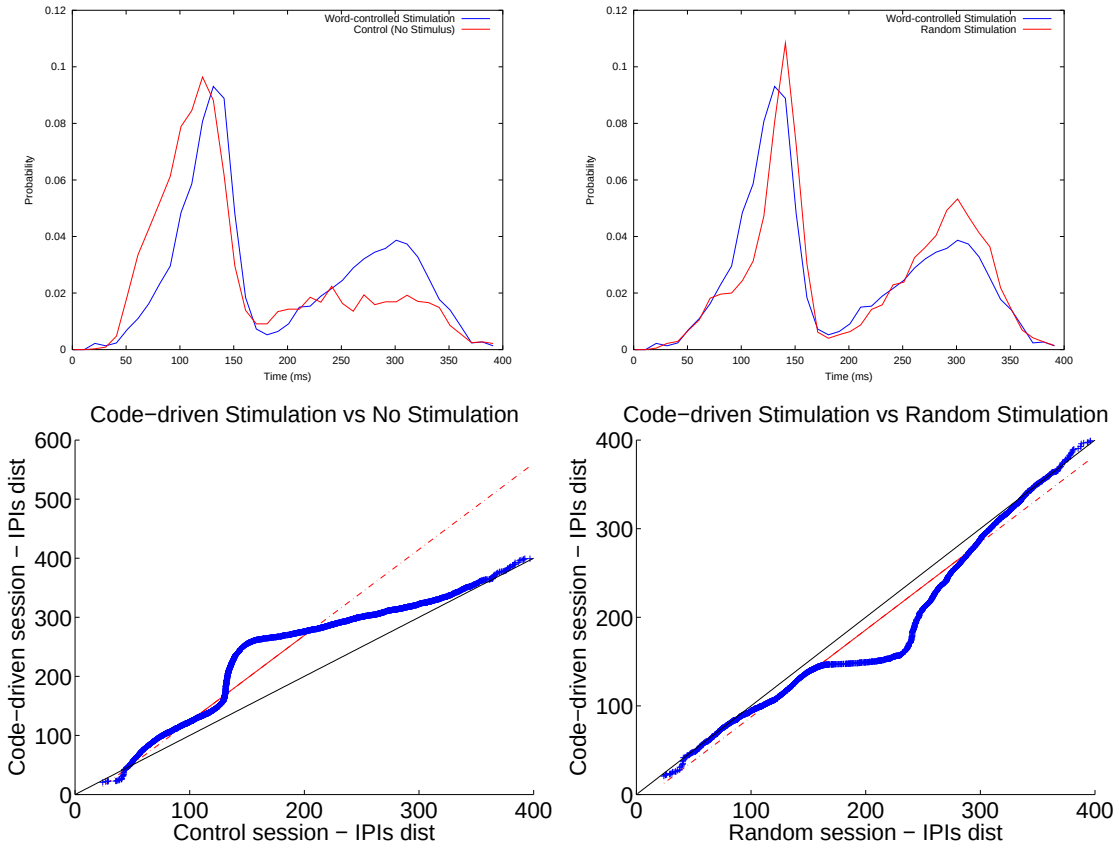

FIGURA 3.6: Distribución de IPIs del experimento 1_sin_80_0101.

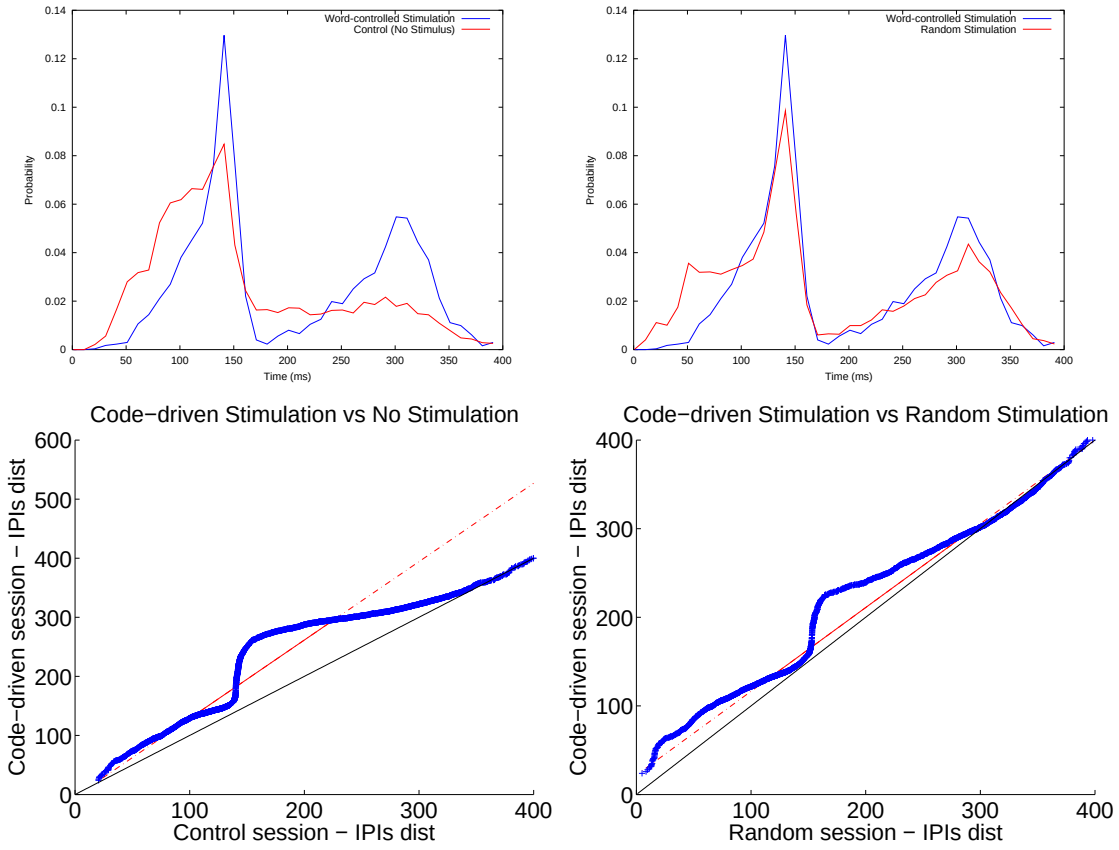

FIGURA 3.7: Distribución de IPIs del experimento 2_sin_80_0101

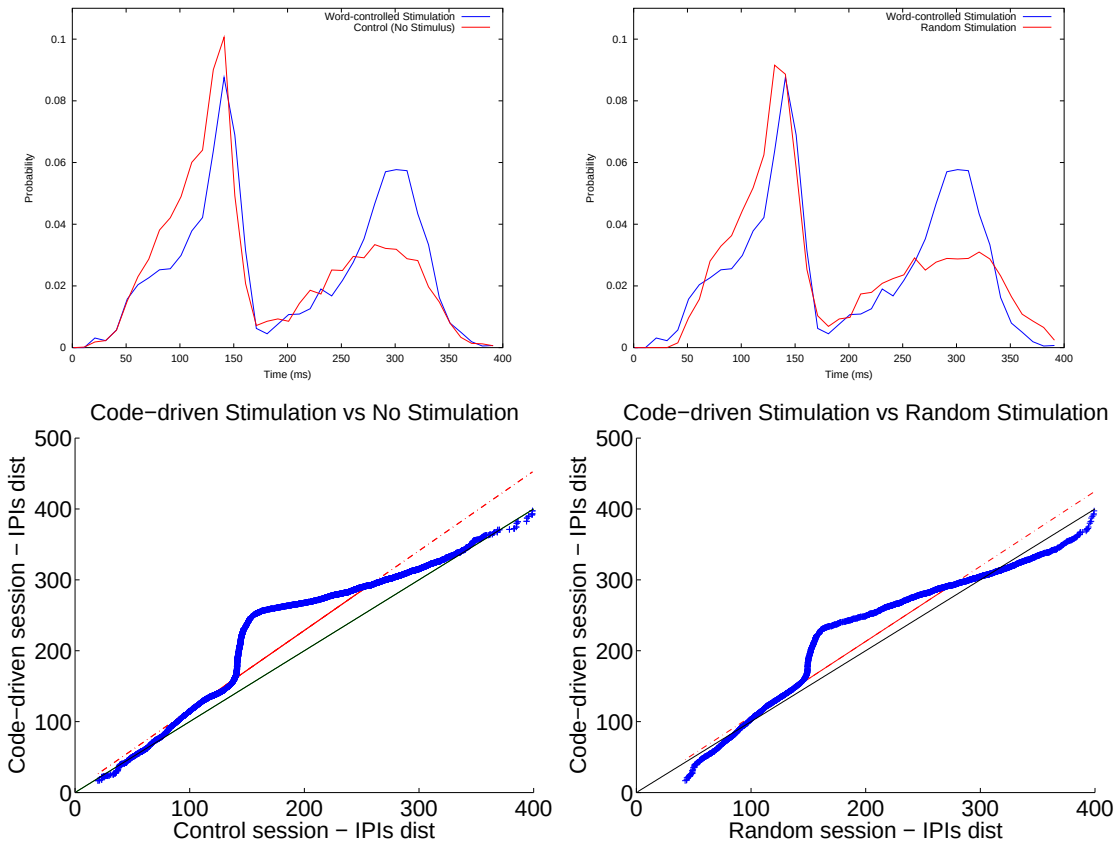

FIGURA 3.8: Distribución de IPIs del experimento 3_sin_80_0101

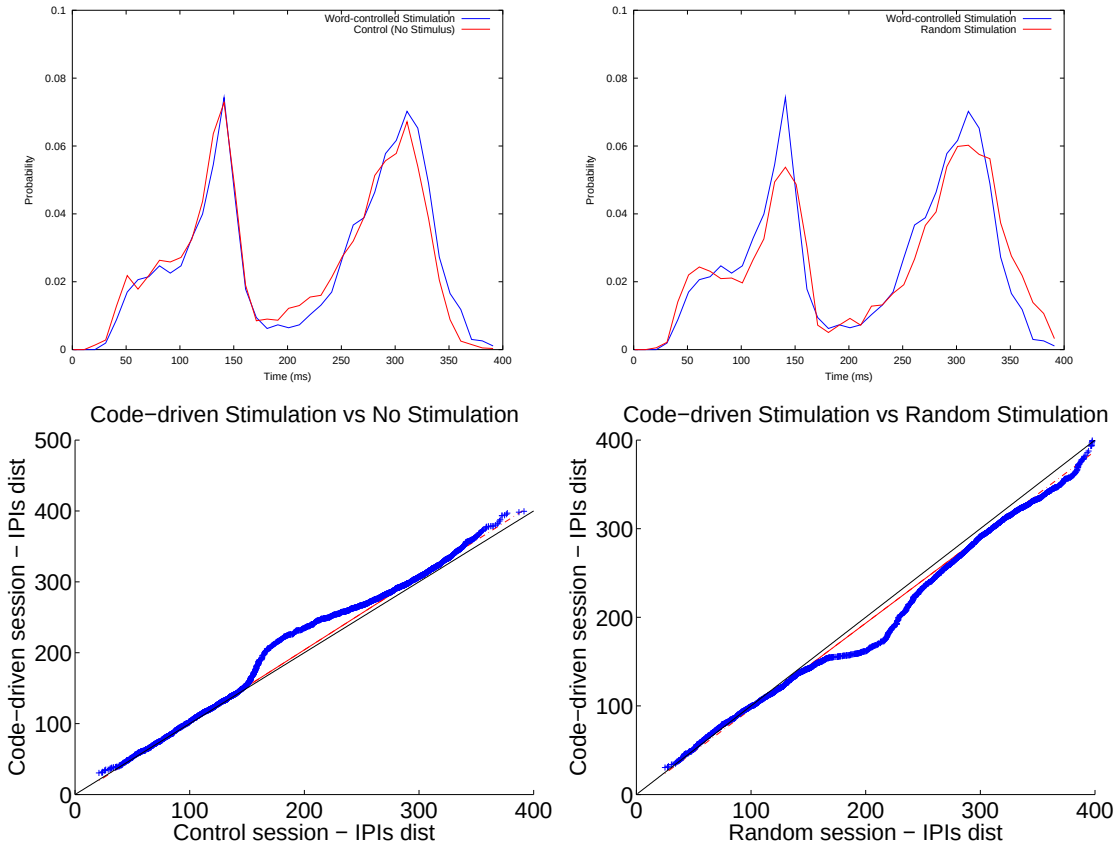

FIGURA 3.9: Distribución de IPIs del experimento 4_sin_80_0101

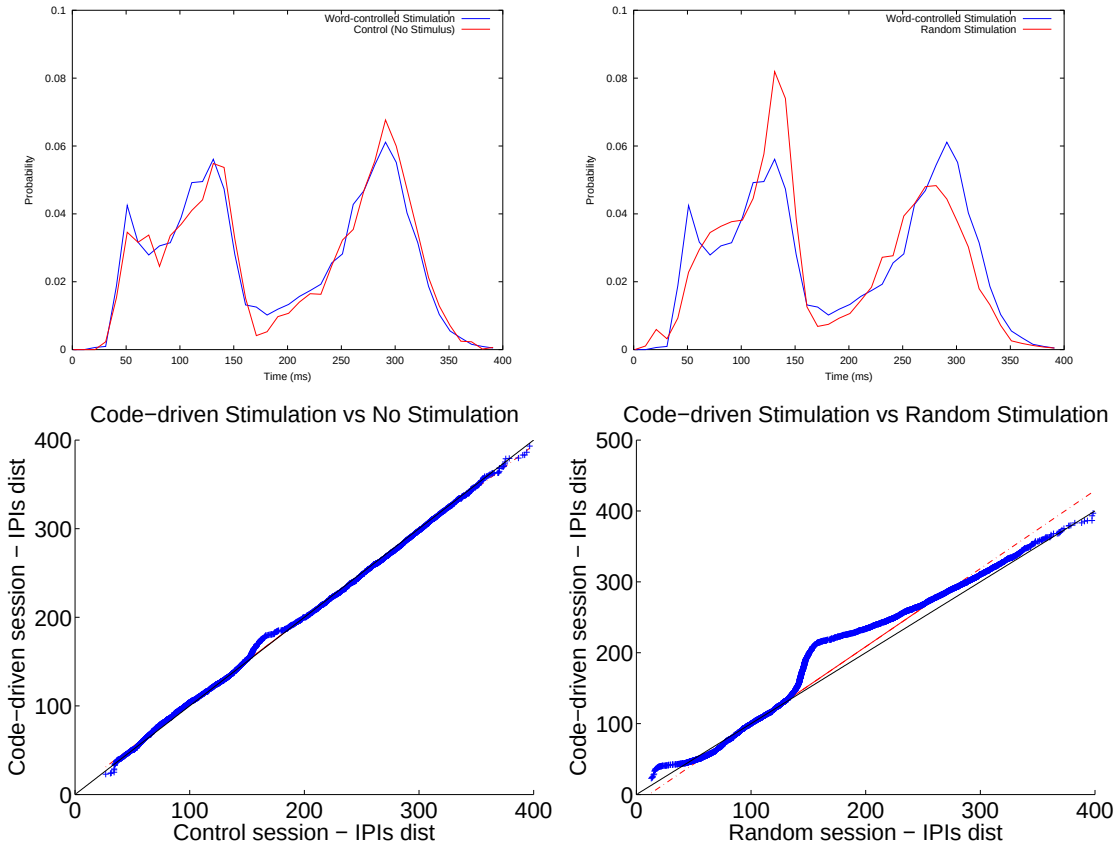

FIGURA 3.10: Distribución de IPIs del experimento 5_sin_110_1001

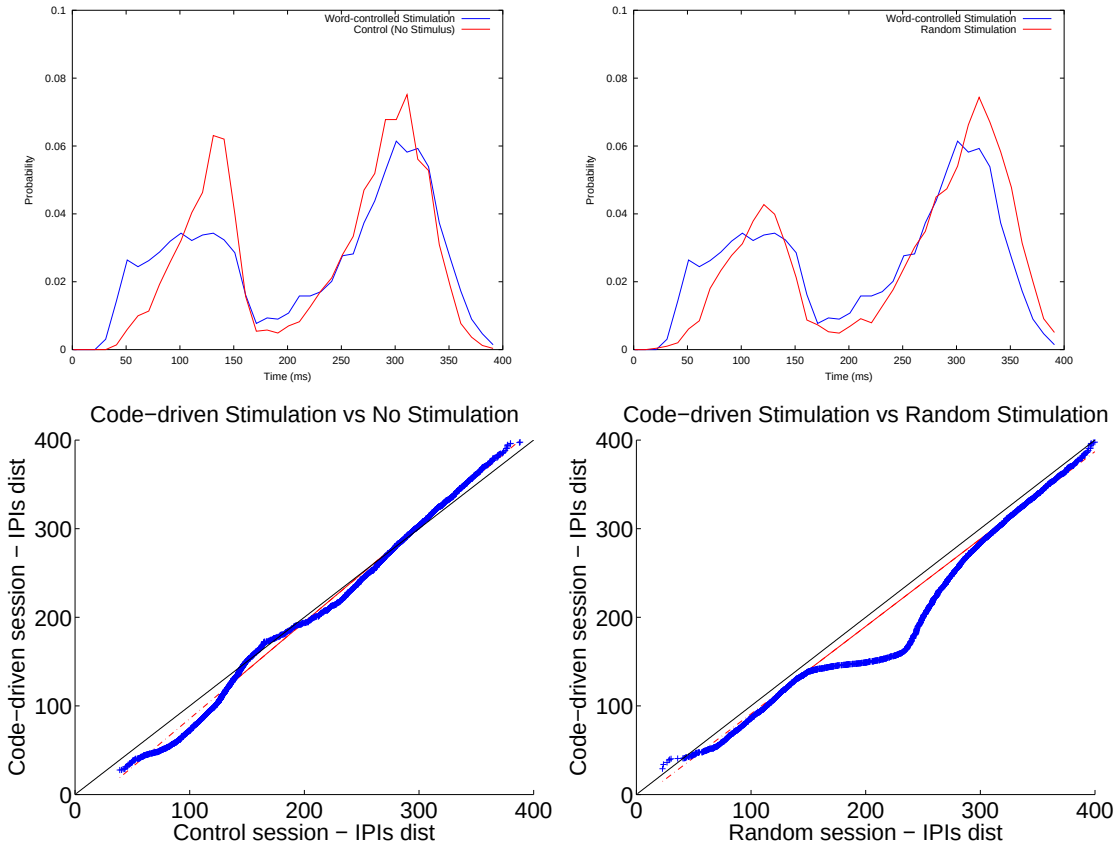

FIGURA 3.11: Distribución de IPIs del experimento 6_sin_110_1001

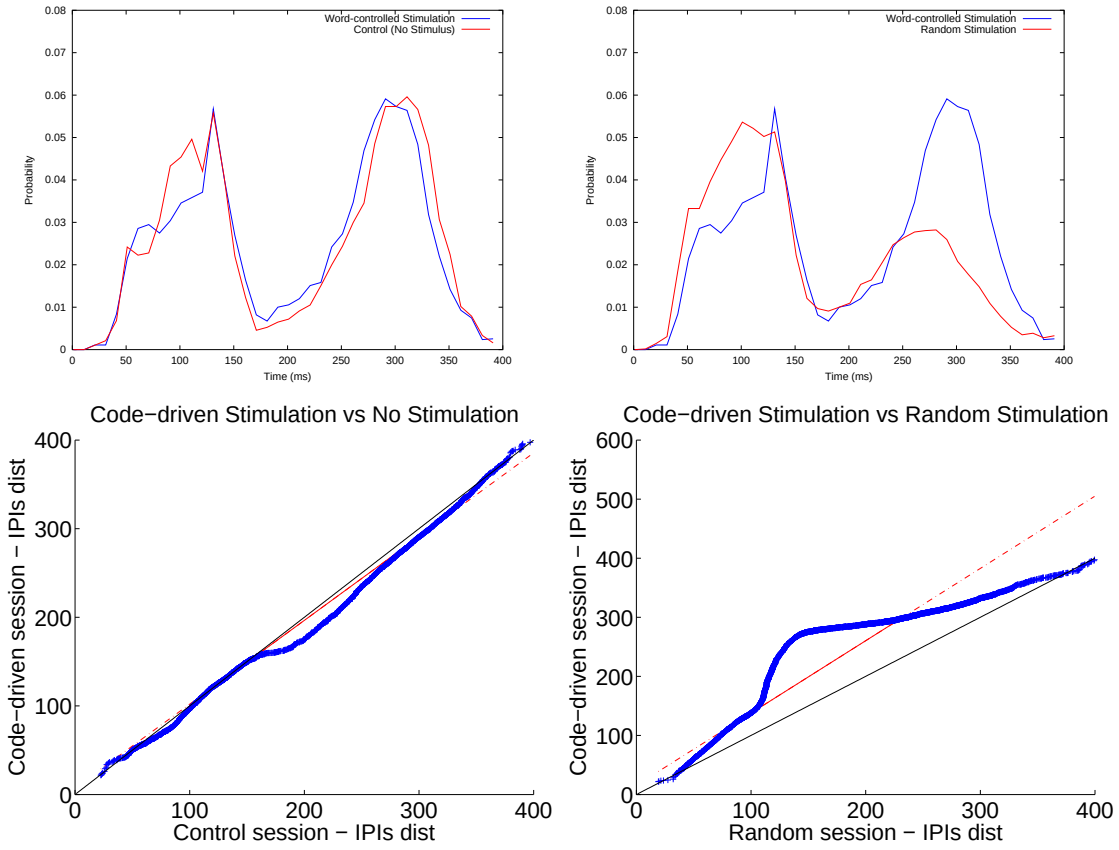

FIGURA 3.12: Distribución de IPIs del experimento 7_sin_110_1001

3.3. Estudio de la influencia del retraso pulso-estímulo en estimulación basada en códigos

3.3.1. Descripción del experimento

Retraso aleatorio en un rango (0-60ms) frente a retraso preciso (10ms).

Como ya se ha explicado en 1.2.1, entendemos por retraso pulso-estímulo el tiempo transcurrido entre el último pulso emitido por el pez previo al lanzamiento de un estímulo artificial y el lanzamiento efectivo de dicho estímulo. La influencia de este retraso pulso-estímulo en la modificación del comportamiento eléctrico del pez ha sido caracterizado en [47].

- Control (10-20 min.)
- Estimulación con retraso aleatorio (30-60 min.)
- Estimulación con retraso preciso de 10 ms. (15-40 min.)

3.3.2. Experimentos realizados

El identificador de experimentos sigue la siguiente codificación:

[número de experimento]_[tiempo de rango maximo en retraso variable]

Código	Figura asociada	Comentario
1_60	3.15	30 min / 15 min
2_60	3.16	30 min / 20 min
3_60	3.17	60 min / 40 min
4_60	3.17	40 min / 20 min

CUADRO 3.3: Listado de experimentos del tipo 3.3 Evolución de la distribución de códigos emitidos y de la entropía en sistemas sin estimulación. En los comentarios se muestra la duración aproximada de cada parte.

3.3.3. Discusión de resultados

En la figura 3.15 podemos ver cómo varía el retraso pulso estímulo en una estimulación dependiente de actividad con un rango de retrasos variable entre 0 y 60 ms respecto

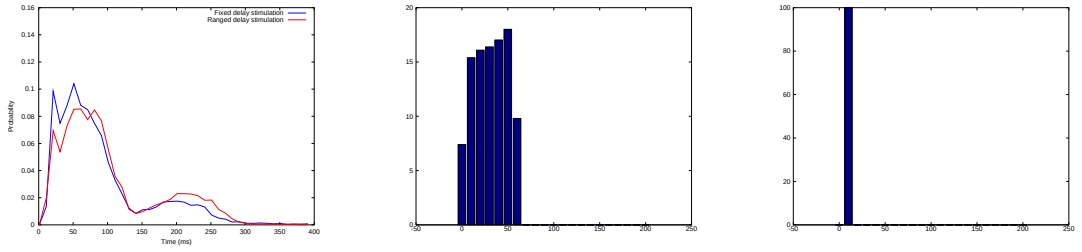

FIGURA 3.15: Resultados experimento retraso fijo y variable 1_60.

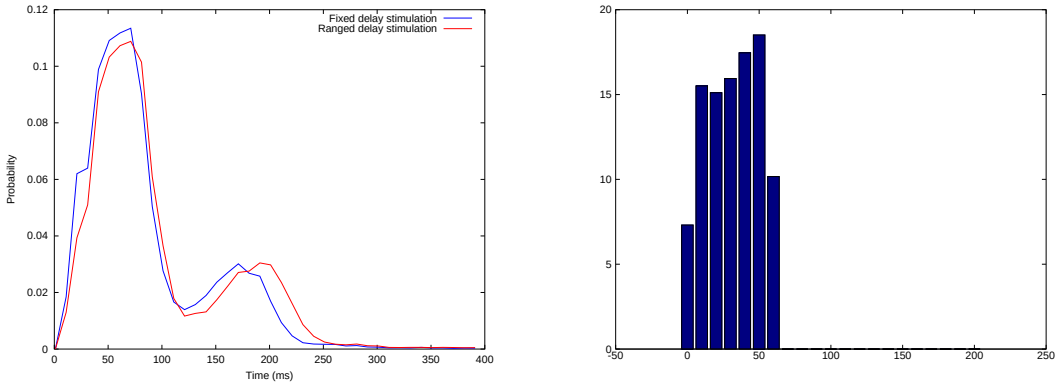

FIGURA 3.16: Resultados experimento retraso fijo y variable 2_60.

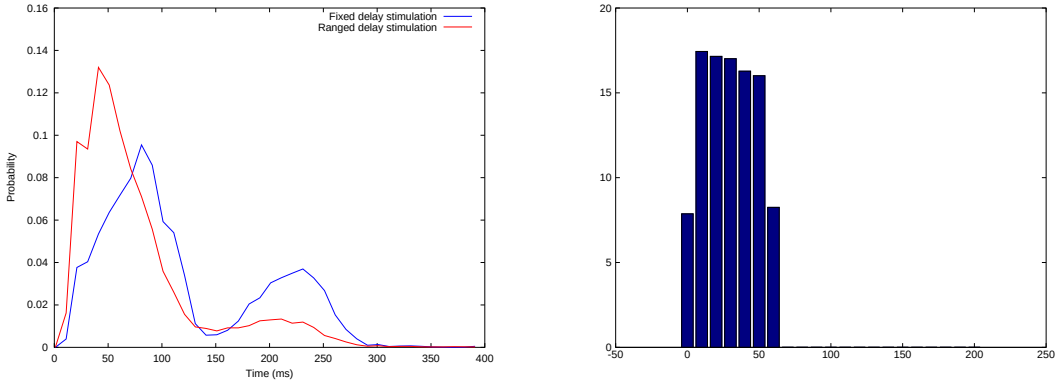

FIGURA 3.17: Resultados experimento retraso fijo y variable 3_60.

de un rango fijo en 10 ms. En el resto de figuras se ha determinado no mostrar la distribución de retrasos para el retraso fijo ya que no aportaba información adicional, al ser constantemente una distribución constante de 10ms.

Es de notar que hay una variación importante en el procesamiento de la información cuando el retraso se presenta de manera fija respecto a cuando este varía en un rango, por mucho que este se encuentre determinado. Se decidió un retraso fijo de 10ms debido a

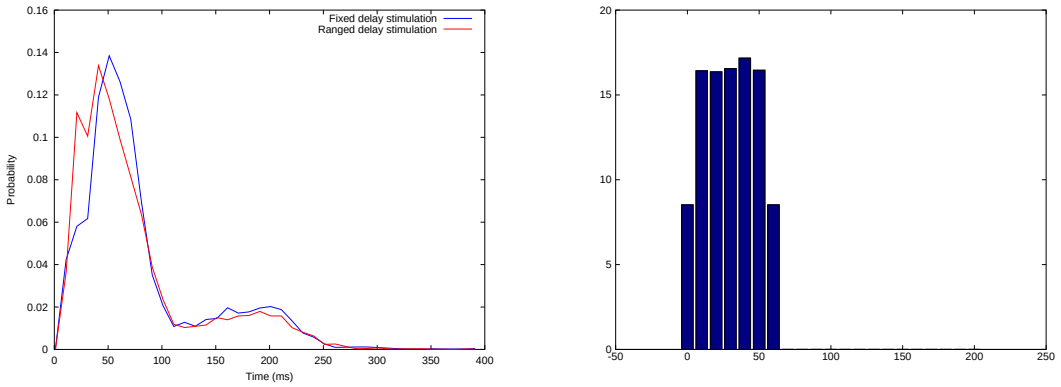

FIGURA 3.18: Resultados experimento retraso fijo y variable 4_60.

que tanto en trabajos previos [47] como en nuestro estudio (sección 3.3) encontramos un importante cambio en el procesamiento de información cuando los estímulos se presentan con dicho retraso. Esta variación es reproducible cuando realizamos experimentos de mayor duración, como puede observarse en la figura 3.17 que muestra los resultados del experimento 3_60. Esto concuerda con los resultados obtenidos en [47], si bien parece que el uso de estímulos de larga duración como los utilizados en nuestro caso no son tan precisos a la hora de variar el comportamiento eléctrico del pez como los estímulos que replican un pulso del propio pez.

El análisis de estos resultados, que corroboran lo mostrado en [47] nos demuestra la importancia de tener en cuenta el retraso pulso-estímulo en nuestros experimentos de cara a evaluar la influencia de la memoria en el sistema biológico bajo estudio. En los experimentos preliminares mostrados anteriormente no se fijó el retraso, por lo que se dificulta la consecución de resultados respecto a los cambios en el procesamiento de información. Por ello, en los experimentos realizados en la sección 3.4 se ha tratado de fijar un retraso de 10 ms con respecto al último pulso al presentar la estimulación.

Debido también a esta necesidad de fijar el retraso pulso-estímulo se ha decidido utilizar palabras terminadas en un bit activo (1). Los códigos terminados en 1 nos permiten realizar una estimulación transcurrido un tiempo fijo con mayor facilidad que los códigos terminados en 0. Esto es debido a que tras la detección del pulso que activa el último bit podemos lanzar la estimulación. De otro modo, para asegurarnos el retraso habríamos de esperar la presencia de un bit, lo que significaría (si tomamos en cuenta la memoria a corto plazo del pez) que estamos estimulando de nuevo tras un código terminado en 1 (el correspondiente al bin que se activa con el último pulso, al que esperamos para fijar el retraso). Es imposible fijar el retraso en 10ms si la palabra termina en 0, ya que en el bin previo a la estimulación no debería haber pulso, lo que se contradice con la presencia de un pulso a 10ms.

3.4. Análisis de la influencia de la memoria a corto plazo en el procesamiento de información

3.4.1. Descripción del experimento

El objetivo de estos experimentos es estudiar la influencia de la memoria en el procesamiento de información en peces eléctricos.. En dichas palabras binarias se codifica la información correspondiente a la memoria a corto plazo. De cara a estudiar la influencia que esta puede tener sobre el estado eléctrico del pez, se va a realizar un estudio basado en estimular al sistema biológico únicamente en función de la emisión de determinados códigos.

Para ello, compararemos el procesamiento de información del pez bajo diferentes condiciones de estimulación. Primero, un control en condiciones normales sin estimulación; segundo, una estimulación dependiente de un evento instantaneo como es la presencia de un solo pulso; tercero y último, una estimulación dependiente de, por un lado, la palabra binaria de dos bits '01' y, por otro, la palabra '11'.

Por tanto, el protocolo para la realización de los experimentos es el siguiente:

- Control sin estimulación (20 min.)
- Estimulación con palabra 01 (40 min.)
- Estimulación con palabra 11 (40 min.)
- Estimulación con pulso (20 min.)

Los tiempos asignados a cada una de las partes son aproximados y en realidad se encuentran limitados a un número de estimulos equivalente a cada parte.

Conste que la elección de esas palabras equivale a dividir en dos el espacio de estados de la estimulación basada en pulso. En el sentido de que la estimulación basada en pulsos va a tener lugar cuando tenga lugar la emisión de cualquiera de las palabras; mientras que en la estimulación basada en palabras se estimula únicamente con una de ellas (y, después, con la otra). La intención es comparar la respuesta cuando la estimulación viene determinada por estados de memoria a corto plazo distintos.

3.4.2. Experimentos realizados

El identificador de experimentos sigue la siguiente codificación:

[número de experimento]_[tipo de estímulo]_[tiempo de bin]

- **Número de experimento:** Por orden temporal, un número único que identifica al experimento
- **Tipo de estímulo:** Forma del estímulo. Se utilizaron dos tipos, seno (sin) y pulso (pulse).
- **Tiempo de bin:** Medido en ms.

Código	Figura asociada	Comentario
1_sin_60	3.19	Orden: Control / 1 / 01 / 11
2_sin_60	3.20	Orden: Control / 11 / 1 / 01
3_sin_60	3.21	Orden: Control / 01 / 11 / 1
4_sin_60	3.22	Orden: Control / 1 / 01 / 11

CUADRO 3.4: Listado de experimentos del tipo 3.3 Evolución de la distribución de códigos emitidos y de la entropía en sistemas sin estimulación. En los comentarios se incluye el orden en que se realizaron las distintas partes del experimento con diferentes condiciones experimentales.

3.4.3. Discusión de resultados

Como puede verse hay una diferencia clara en el procesamiento de información bajo diferentes condiciones experimentales. Si miramos a los histogramas de IPIs vemos un cambio claro entre las curvas resultantes de las distintas distribuciones de IPIs bajo condiciones experimentales distintas. En concreto, la figura 3.20 da una muestra muy clara de que la variación que ocurre con 01 difiere considerablemente respecto a la que tiene lugar con 1 o con 11. Además, también vemos desplazamientos entre los histogramas de IPIs como resultado de establecer un ciclo cerrado guiado por eventos instantaneos (1) o por códigos de al menos dos bits que tienen en cuenta la importancia de la memoria. En el caso de la diferencia de la estimulación establecida con el código 01, tanto con respecto al control como con respecto a la estimulación con 1 es muy clara en la figura 3.20. Pero si atendemos a la estimulación con 11 respecto a la que tiene lugar como resultado del evento instantaneo 1 también vemos una tendencia, en el caso experimental de la presencia de un ciclo cerrado estimulando con la palabra 11, a disparar con frecuencias más altas. Conclusiones similares pueden extraerse del resto de graficas.

Por otro lado, si atendemos a los QQPlots vemos que la interpolación lineal de los datos no seguiría una recta de inclinación de 45º (lo que indicaría una distribución equivalente

entre los dos conjuntos de datos) por lo que las distintas condiciones de estimulación hacen variar de manera no lineal las distribuciones de IPIs con respecto al control. Por otro lado, la estimulación que responde únicamente a eventos instantaneos y no toma en cuenta la memoria tiene una distribución diferenciada con respecto a la que se encuentra guiada por códigos de 2 bits. Un ejemplo es la figura 3.19, donde el distanciamiento de la recta intercuantílica es mucho mayor en el caso de la estimulación dirigida por eventos instantaneos, especialmente para IPIs mayores de 200ms. Este resultado con respecto a la variación en IPIs por encima de 200ms se repite en la figura 3.20 y también puede observarse en la figura 3.22.

Aunque el cambio en el procesamiento de información no sea siempre el mismo, sí puede derivarse que el cambio resultante de la estimulación basada en códigos binarios de 2 bits tiene una influencia distinta sobre el sistema que aquella estimulación que depende únicamente de eventos instantaneos que no tienen en cuenta la influencia de la memoria. Es necesario tener en cuenta que al tratarse de un sistema biológico, el comportamiento eléctrico del pez depende no únicamente de la estimulación eléctrica artificial y controlada que emitimos. A pesar de que se ha intentado en lo posible aislar al sistema biológico, el aislamiento no es total, por lo que hay una influencia externa de otros estímulos, así como del estado previo del pez y otros condicionantes.

Pese a todo, una variación como la mostrada da indicios importantes de la influencia que tiene la memoria a corto plazo en el procesamiento de información, ya que a pesar de mantener la densidad de estímulos emitidos al pez, los retrasos fijos y el tipo y la cantidad de estímulos, seguimos observando cambios considerables y hasta cierto punto reproducibles en los experimentos realizados.

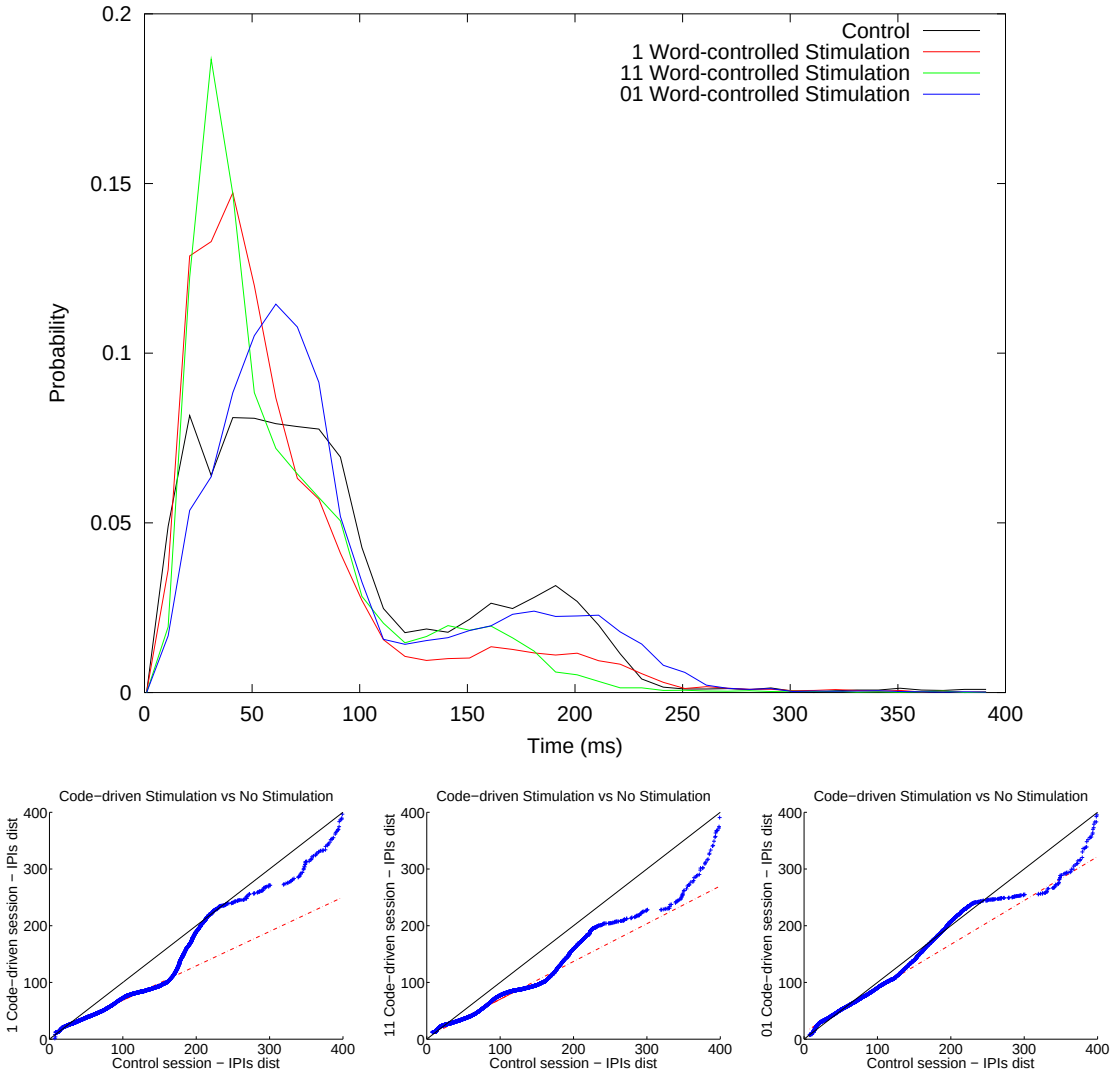

FIGURA 3.19: Histograma de IPIs y QQPlots como resultado del experimento 1_sin_60.

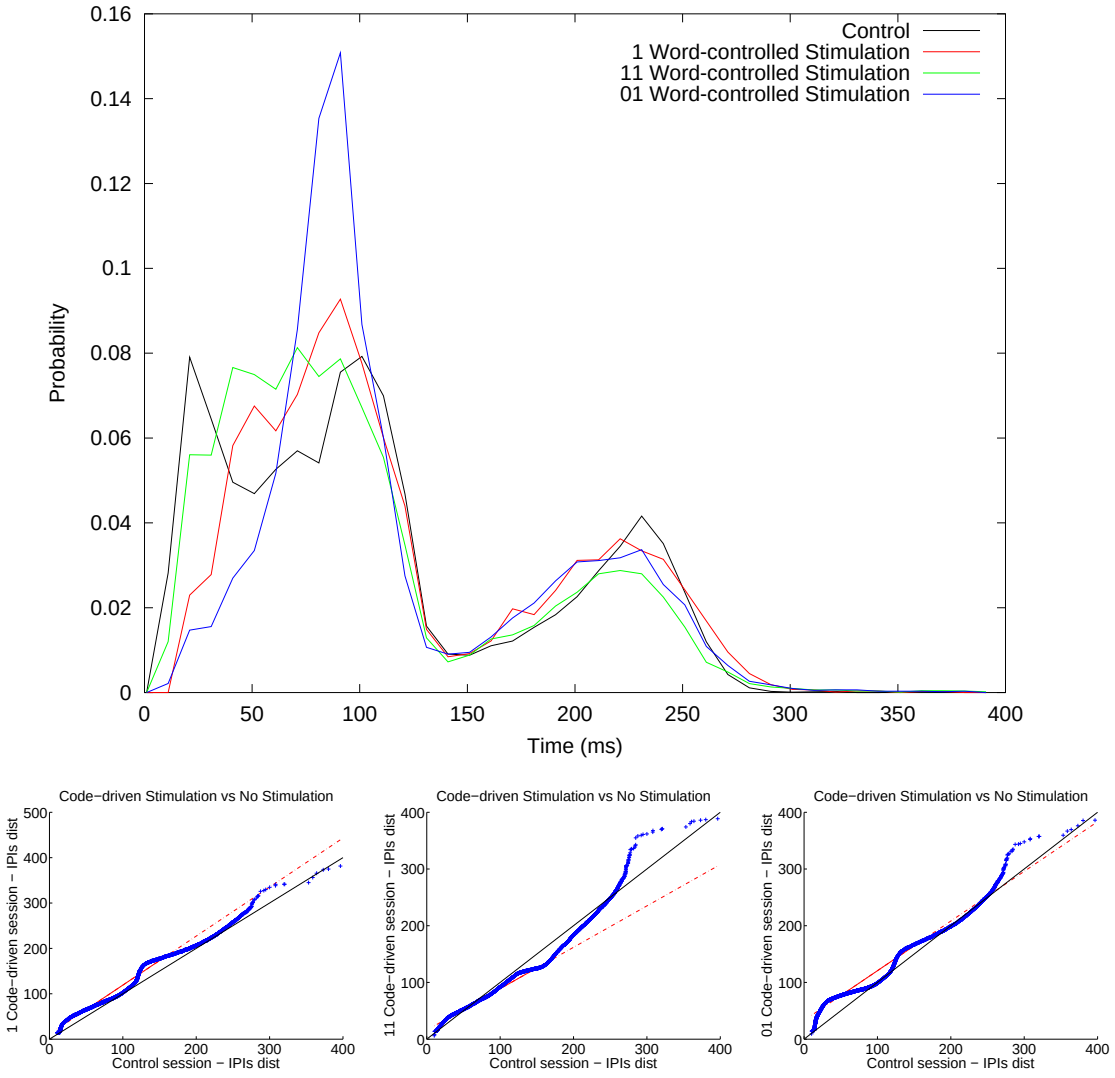

FIGURA 3.20: Histograma de IPIs y QQPlots como resultado del experimento 2_sin_60.

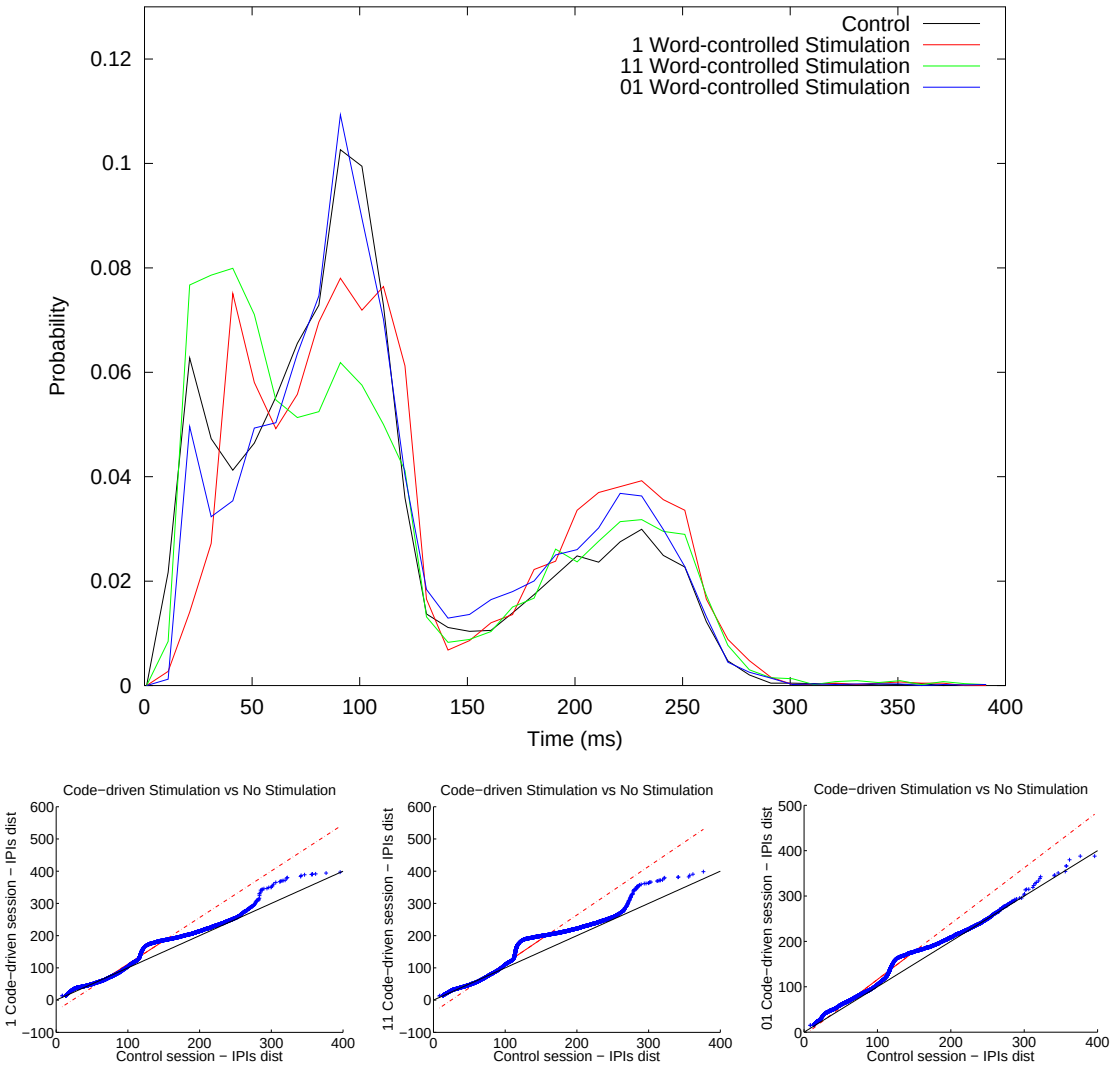

FIGURA 3.21: Histograma de IPIs y QQPlots como resultado del experimento 3_sin_60.

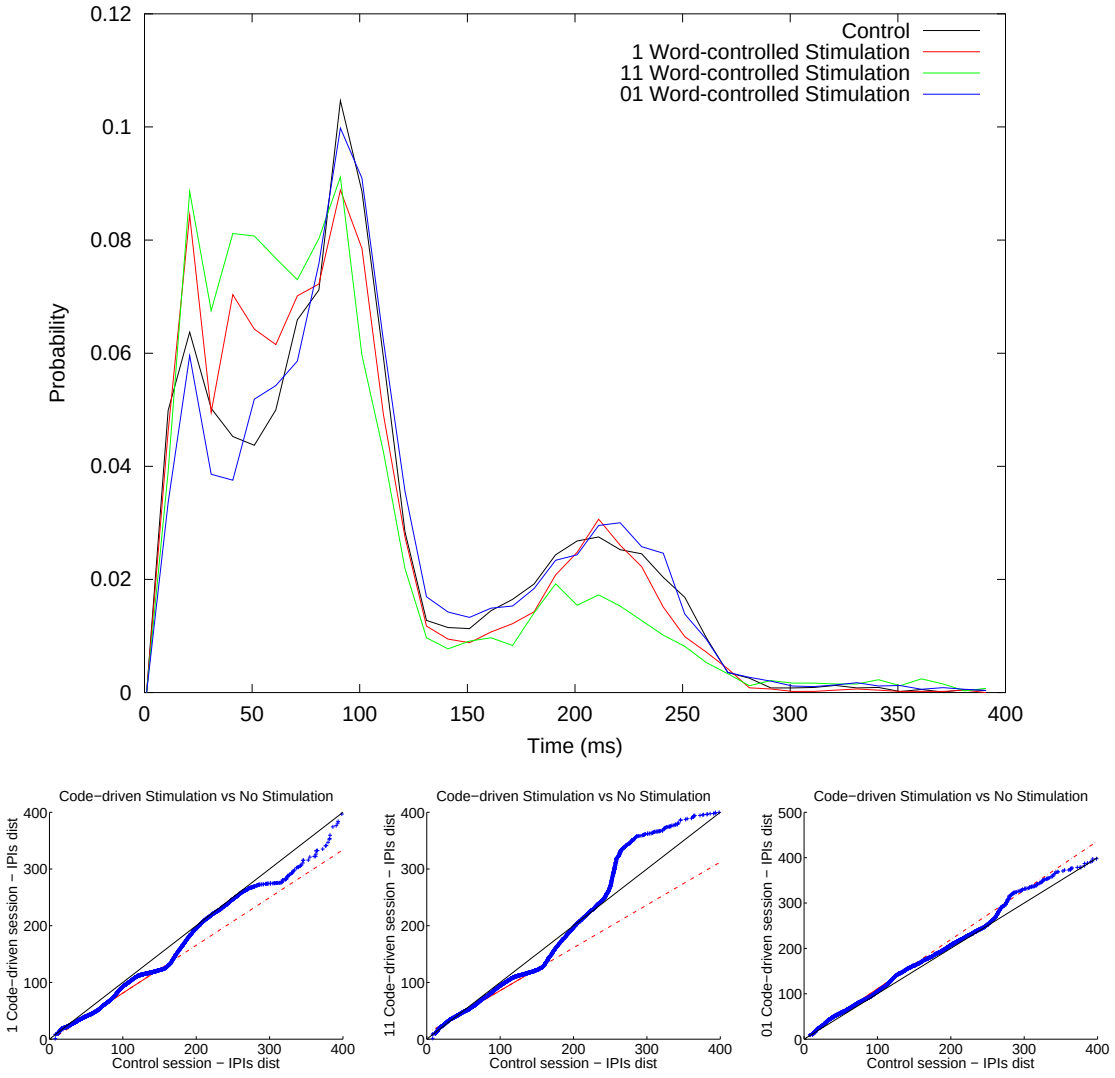

FIGURA 3.22: Histograma de IPIs y QQPlots como resultado del experimento 4_sin_60.

Capítulo 4

Conclusiones

El estudio del procesamiento de información en procesos comunicativos naturales puede tener importantes aplicaciones en los campos de la ingeniería y las telecomunicaciones.

El uso de ciclos cerrados de comunicación y estimulación bidireccional permite revelar dinámicas ocultas bajo protocolos de estimulación tradicionales, basados en estimulos completamente artificiales o la emisión de estímulos pregrabados.

Resulta necesario el uso de tecnologías de tiempo real para establecer ciclos cerrados en sistemas biológicos sensibles a variaciones leves del retraso en la recepción de estímulos de respuesta.

El uso de herramientas de teoría de la información y otros métodos estadísticos nos permite caracterizar y estudiar el estado eléctrico del pez de cara a entender las variaciones que realiza en el procesamiento de la información.

La plataforma hardware-software utilizada en este proyecto y que hace uso de estas tecnologías hace posible la caracterización del sistema biológico ya que permitir la adquisición de datos en vivo durante periodos largos de tiempo, como los que en este proyecto se realizan en la caracterización entrópica.

Respecto al estudio de la evolución de la superficie entropica en periodos largos de tiempo sin estimulación, hemos encontrado que, en ocasiones determinadas, existen variaciones importantes de la entropía máxima y media. Esto justifica un control entrópico previo a cada experimento que ayude a caracterizar el estado del pez.

En relación con los estudios preliminares de la influencia de la memoria, hemos descubierto cierta variabilidad que resulta dificil de reproducir. En primer lugar debido a la alta variabilidad de los sistemas biológicos ya que se trata de un sistema dinámico en que resulta determinante el estado previo del mismo para poder reproducir un resultado

ante un protocolo de estimulación equivalente. También debido a que no se ha tenido en cuenta la influencia de fijar el retraso pulso-estímulo. Con todo, existen cambios que muestran una cierta reproducibilidad en el procesamiento de información para una estimulación guiada por un mismo código, tal como podía observarse en las figuras 3.7 y 3.8.

En la serie de experimentos sobre el retraso pulso-estímulo se ha ratificado la hipótesis de [47] sobre la influencia de un retraso fijo de 10ms.

Finalmente, el estudio final sobre la influencia de la memoria muestra resultados prometedores, ya que descubrimos una variación en presencia de estimulación basada en uno de los códigos que no se reproduce cuando la estimulación depende únicamente del evento instantaneo que finaliza dicho código. Esta variación se mantiene a pesar de mantener los parámetros de: retraso pulso-estímulo y número de estimulaciones; lo que parece indicar que existe una influencia de la memoria sobre el propio comportamiento eléctrico en los procesos que guían el tratamiento de la información.

4.1. Consecución de objetivos

1. El objetivo general de estudiar el modo en que la información se procesa se ha desarrollado a lo largo del proyecto mediante la consecución de los subobjetivos

    a) Respecto al estudio de la memoria, se han logrado indicios prometedores sobre la influencia de la memoria, que han quedado descritos en la sección 3.4

        1) Los mecanismos de estimulación en ciclo cerrado guiados por códigos (códigos que portan información sobre la memoria del pez) se han implementado sobre la plataforma de ADClamp para operar en tiempo real dentro del módulo words, por lo que este objetivo se ha cumplido plenamente.

        2) Del mismo modo, se ha concluido la implementación de un protocolo de digitalización y binarización de la señal, tanto para operar online como offline.

        3) La selección de los parámetros de digitalización y binarización por un criterio de máxima entropía ha sido posible gracias al desarrollo de aplicaciones para el calculo de la superficie entrópica, por lo que se considera el objetivo como completado.

        4) Por último, con el desarrollo del proyecto se han refinado los protocolos experimentales iniciales para el estudio de la memoria, tomando en cuenta la influencia del retraso pulso-estímulo, del estado previo (respecto a la

distribución de IPIs y a la entropía de palabras binarias) y del número de estímulos. Con estos avances se han logrado los resultados preliminares, bastante prometedores, que se muestran en 3.4, por lo que podemos decir que el objetivo se ha cumplido en su mayor parte.

b) Se ha cumplido el objetivo de analizar la capacidad de transmisión de información de la señal eléctrica y la evolución temporal de dicha capacidad. Este estudio se ha realizado en 3.1.

1) Con el desarrollo de aplicaciones para el cálculo y la visualización de la entropía, se ha cumplimentado el objetivo del cálculo de la capacidad de transmisión de información de la señal emitida por el pez.

2) También se han implementado las herramientas de análisis de evolución de la entropía a lo largo del tiempo.

c) Se ha tomado en cuenta la importancia del retraso post-pulso en la recepción de estímulos en el desarrollo de protocolos experimentales para estudiar el problema de la memoria.

1) Las aplicaciones desarrolladas para la estimulación se han implementado sobre una plataforma en tiempo real para asegurar de manera estricta los requisitos temporales.

2. Respecto al desarrollo de metodologías de análisis del procesamiento de información que puedan aplicarse al estudio de otros sistemas biológicos, la metodología aplicada en este proyecto podría aplicarse a señales biológicas en forma de pulsos que codifiquen información en la frecuencia de emisión de tales pulsos. Podría ser el caso, por ejemplo, de las neuronas. Si bien esta idea y otras aplicaciones han estado presentes a lo largo del desarrollo del proyecto, no se trata de un objetivo que se haya cumplido efectivamente.

3. Lo mismo ocurre con respecto a la implementación de protocolos de estimulación en ciclo cerrado que puedan aplicarse al estudio de otros sistemas biológicos. El protocolo de codificación/binarización también puede aplicarse a otros sistemas de procesamiento de información con señal en pulsos.

4. Finalmente, el desarrollo de aplicaciones en tiempo real para su aplicación al estudio de otros sistemas biológicos se ha cumplido en tanto en cuanto las aplicaciones ha sido desarrolladas, pero no se ha probado su aplicación a otros sistemas biológicos.

4.2. Trabajo futuro

Como trabajo futuro:

La implementación en la arquitectura HW-SW de la posibilidad de un registro para analizar la comunicación entre dos peces reales, de modo que además nos permita modificar y filtrar en tiempo real la posibilidad de modificar y filtrar los mensajes que se envían entre ellos.

La utilización de teoría de control y, más concretamente, de filtros de Kalman para controlar el estado del pez, tomando a este como un sistema dinámico.

El análisis de la información mutua entre la señal adquirida del pez y la señal estimuladora bajo dferentes condiciones de estimulación en los experimentos.

Un estudio continuado sobre la distribución de palabras emitidas por el pez (y la entropía resultante) durante su periodo vital podría darnos información sobre cómo evoluciona esta en el largo plazo y permitir caracterizar diferentes estados del sistema a lo largo de su vida. Un sistema de adquisición de la señal del pez como el que se utiliza en este proyecto está especialmente indicado para experimentos de este tipo, ya que permite un registro no invasivo de la actividad eléctrica del pez que podría mantenerse durante periodos de tiempo muy largos.

Apéndice A

Real Time Application Interface (RTAI)

Real Time Application Interface (RTAI, Interfaz para Aplicaciones en Tiempo Real) es una implementación de Linux para tiempo real basada en un principio en RTLinux, y actualmente en ADEOS. Se basa en el núcleo Linux, proporcionando la capacidad de hacerlo completamente requisable (preemptable), lo que permite determinar los tiempos de respuesta.

Linux no es un sistema en tiempo real por varias razones:

- Sincronización de grano grueso, las interrupciones del sistema no pueden conocerse de antemano y atiende a estas antes que a otros procesos, por lo que no puede determinarse con certeza el tiempo de respuesta de las aplicaciones.
- El planificador trata de ser justo, dando ranuras de tiempo en el procesador a todas las aplicaciones, lo que agrava la imposibilidad de determinar los tiempos de respuesta.
- Paginación de memoria, no tenemos forma de saber cuánto tarda este proceso ni de preveerlo.
- Reordenación de las tareas (para optimizar el uso del HW) imposibilitan que los tiempos de respuesta sean determinados. Batching y otros.
- Es relativamente sencillo realizar un experimento que demuestre como la latencia del sistema Linux varía en función de las aplicaciones que se encuentran abiertas y otras variables mencionadas.

La figura A.1 muestra el funcionamiento de un sistema Linux sin RTAI, dividido por niveles. Las interrupciones hardware son manejadas por las tareas en el espacio del kernel y son estas las que se comunican directamente con el HW. El espacio de usuario se comunica con el Kernel mediante llamadas al sistema.

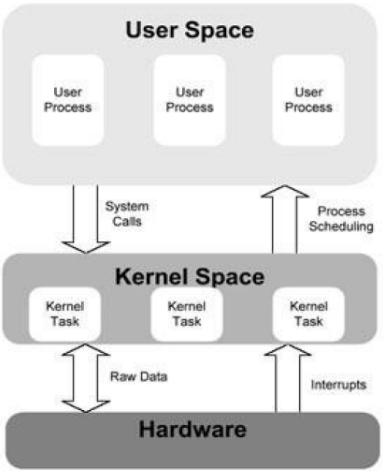

FIGURA A.1: Comunicación Kernel con espacio Usuario en RTAI

.

El funcionamiento de RTAI es el siguiente: RTAI añade un pequeño núcleo Linux de tiempo real bajo el núcleo estándar de linux y trata al núcleo linux como una tarea de menor prioridad. RTAI además proporciona una amplia selección de mecanismos de comunicación entre procesos y otros servicios de tiempo real. Puede verse la arquitectura de RTAI en la Figura A.2.

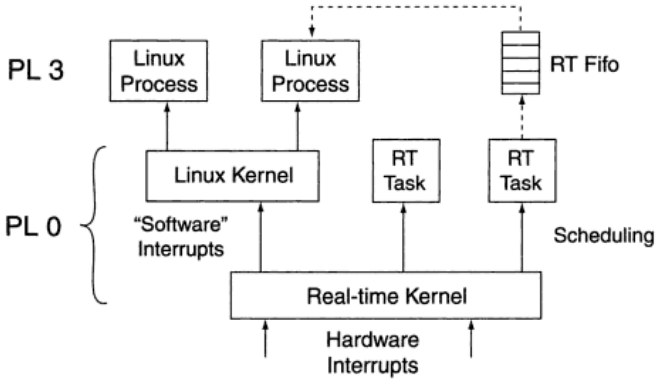

FIGURA A.2: Esquema de la arquitectura RTAI. El kernel de Linux se encuentra al mismo nivel que las aplicaciones tiempo real. La comunicación entre procesos linux y tareas en tiempo real se realiza mediante colas FIFO (o memoria compartida.

Las comunicaciones entre las tareas RTAI, el núcleo standar de linux y las aplicaciones de usuario se realiza mediante memoria compartida, colas FIFO e interrupciones.

A.1. Características RTAI

A.1.1. Comunicación y sincronización entre tareas kernel

Algunos de los mecanismos que RTAI nos facilita para gestionar la comunicación y sincronización entre tareas del Kernel son los siguientes:

- Semáforos: Semáforos convencionales con operaciones signal y wait.
- Mailboxes: Colas que se crean con una tamaño arbitrario y cantidades arbitrarias de datos pueden ser escritos y leidos de una mailbox.
- Mensajería: Permiten la comunicación directa de tarea a tarea. Una tarea puede enviar un simple dato directamente a otra. La tarea receptora puede esperar por un mensaje desde una tarea origen específica o desde cualquier tarea. También hay una versión full duplex de mensajería llamada "Remote Procedure Calls".

Todos estos mecanismos se implementan en el módulo rtai_sched.

A.1.2. Comunicación con procesos Linux

RTAI también implementa mecanismos para permitir la comunicación entre tareas tiempo real y aplicaciones de usuario: RT FIFOs y Memoria compartida.

RT FIFOs: Son conexiones punto a punto entre un proceso linux y una tarea en tiempo real. Normalmente se utilizan de modo unidireccional, por lo que para la comunicación en los dos sentidos se utilizan dos FIFOs, una para enviar los comandos y otra para leer las respuestas. Desde espacio de usuario se tratan las RT FIFOs como dispositivos de caracteres (/dev/rtf0 a /dev/rtf63). Un proceso abre una FIFO como si se tratase de un archivo y la lo utiliza para leer o escribir usando respectivamente read() o write() sobre el descriptor del archivo. Los dispositivos rtf se crean automáticamente al instalar RTAI.

Las tareas en tiempo real acceden a la FIFO a través de la API específica de RTAI (que veremos más adelante con más detalle).

Memoria compartida: El uso de RT FIFOs tiene sentido para pequeñas cantidades de información a transmitir entre una tarea tiempo real y un proceso linux. Cuando la comunicación es de una tarea tiempo real a varias aplicaciones de usuario que leen los datos que genera esta, tiene sentido el uso de memoria compartida. La tarea en tiempo real escribe en la memoria compartida y las aplicaciones pueden leer de ella libremente.

A.1.3. Planificación Periódica vs Planificación One-shot

RTAI soporta los dos modelos de interrupción de reloj. En modo periódico, estableces el período con el que el reloj mandará interrupciones. Así es como funcionan la mayoría de sistemas operativos. El problema es que las tareas periódicas solo pueden ejecutarse en incrementos de tiempo dados por el intervalo de interrupción. Si el intervalo es grande, el periodo más pequeño para una tarea es grande, lo que lleva a desperdiciar tiempo. Si el intervalo es pequeño, hay más interrupciones y el tiempo dedicado a servir las interrupciones del reloj aumenta, lo que también desperdicia tiempo.

El modo One-shot de RTAI es una solución a esto. En este modo se reprograma el reloj cada vez que se sirve una interrupción, se calcula el tiempo hasta que se produzca el siguiente evento y se establece ese tiempo como el intervalo hasta la siguiente interrupción. De tal modo que el reloj deja de ser periódico. El problema de este acercamiento es el tiempo perdido en el cálculo de los intervalos.

Puede establecerse dinámicamente un modo u otro mediante:

Por defecto se utiliza el modo periódico y es el utilizado en ADClamp.

Apéndice B

Words Module

El módulo Words de ADClamp es el encargado de realizar la binarización de la señal electrorreceptiva emitida por el pez eléctrico, de generar de manera online los histogramas de emisión de palabras, de establecer el ciclo cerrado dirigido por la emisión de códigos binarios y de implementar el protocolo de estimulación aleatorio.

Este módulo está dividido en dos espacios, parte del módulo trabaja en espacio de usuario, como una aplicación cualquiera su ejecución está determinada por la asignación de tiempos que realiza el sistema operativo; otra parte opera como una tarea en tiempo real de RTAI y sus interrupciones tienen prioridad sobre las del SO. Por tanto, los códigos de la sección B.1 están escritos en C con el objetivo de ser cargados como módulos del espacio de kernel de RTAI.

Más información sobre los protocolos de estimulación basados en códigos puede encontrarse en la sección 2.2.3, en el capítulo sobre metodología. También se describe el funcionamiento del módulo Words en la sección sobre arquitectura software, subsección 2.6.1.

A continuación se lista el código utilizado en el módulo.

B.1. Módulo Words como tarea en tiempo real

B.1.1. Words_shm

```
#ifndef __WOR_SHM__
#define __WOR_SHM__

#include "../rt/words/wordsBuffer.h"
```

```
#define WORDSSHM "wordshm"
#define MAX_BITS_WORD 10
#define MAX_WORDS 1024

typedef struct
{
    char activated[3];  //[0] - WordsDetector
                                                    //[1] - WordsHistogram
                                                    //[2] - WordsInformation

    double threshold;
    double bitTime;

    int wordSize;
    char word[MAX_BITS_WORD];
    int numWords;
    double windowTime;
    double refractTime;

        int hist[MAX_WORDS];
        unsigned char histSem;

    int channel_in;

    double randomPeriod;
} WORDS_SHM;

#endif
```

B.1.2. Generador de histograma de palabras (Words Histogram Generator)

```
double histogramModel(double currentV, double time){
    static double lastTime, initTime;
    static short numCalls = FIRST_CALL;
    static char out = BIT_NOT_DETECTED_OUT;
    static double bitTime;
    static WordsBuffer *wb;
        static int n=0;

    if (numCalls==FIRST_CALL){
```

```
        wb = &wordsbuf;
        wbInit(wb,wordsShm->wordSize, wordsShm->numWords);
        initTime = time;
        numCalls = NEW_TIME_WINDOW;
    }

    if (time-initTime < (wordsShm->windowTime*1000)){
        if (numCalls==NEW_TIME_WINDOW){ //New time window
            n++;
            lastTime = time;
            bitTime = wordsShm->bitTime;
            numCalls++;
        }

        if ((detect_spike(currentV, time))&&(out != BIT_DETECTED_OUT)){
    //DETECT SPIKE
            out = BIT_DETECTED_OUT;
            wbBitInsert(wb,1); //Store 1 for this window time
            wbStoreWord(wb);
        }

        if (time >= initTime+n*wordsShm->bitTime){ //Window time ended
            if (out == BIT_NOT_DETECTED_OUT){
                wbBitInsert(wb,0); //Store 0 for this window time
                wbStoreWord(wb);
            }
            out = BIT_NOT_DETECTED_OUT;
            numCalls = NEW_TIME_WINDOW;
            return END_WINDOW_OUT;
        }
    }
    else {
        initTime = time;
        wbCreateHistogram(wb, wordsShm->hist, wordsShm->numWords);
        wordsShm->histSem++;
        n=0;
        return END_HISTOGRAM_TIME;
    }

    return out;
}
```

B.1.3. Detector de palabra binaria o código para estimulación en ciclo cerrado

```
double detectorModel(double currentV, double time)
{
    static double lastTime;
    static short numCalls = FIRST_CALL;
    static char out = BIT_NOT_DETECTED_OUT, detectedWord = FALSE;
    static double bitTime, wordDetectTime=100000, stimTime= -400;
    static WordsBuffer *wb;

    if (numCalls==FIRST_CALL){
        //wordsShm->refractTime = 200;
                wb = &wordsbuf;
        wbInit(wb,wordsShm->wordSize, wordsShm->numWords);
        numCalls = NEW_TIME_WINDOW;
    }

    if (numCalls==NEW_TIME_WINDOW){ //New time window
        lastTime = time;
        bitTime = wordsShm->bitTime;
        numCalls++;
    }

    if ((detect_spike(currentV, time))&&(out != 1)){ //DETECT SPIKE
        out = BIT_DETECTED_OUT;
        wbBitInsert(wb,out); //Store 1 for this window time
        if (-1!=wbStoreWord(wb)){
            detectedWord = wbCheckWordMatch(wb, wordsShm->word);
            if (detectedWord) wordDetectTime=time;
                }
    }

    if (detectedWord && time>=wordDetectTime+wordsShm->minDelay){ //precise
    delay
        if (stimulator->trigger_function){
           #ifdef __DEBUG__
            rt_printk ("\nStimulator trigger_function");
            #endif
            stimulator->trigger_function();
            stimTime = time;
        }
        detectedWord = FALSE;
        return WORD_DETECTED_OUT;
```

```
    }

    if (bitTime < time-lastTime){ //Window time ended
        if (out == BIT_NOT_DETECTED_OUT){
            wbBitInsert(wb,out); //Store 0 for this window time
            if (-1!=wbStoreWord(wb)){
                detectedWord = wbCheckWordMatch(wb, wordsShm->word);
                if (detectedWord) wordDetectTime=time;
            }
        }

        out = BIT_NOT_DETECTED_OUT;
        numCalls = NEW_TIME_WINDOW;

        return END_WINDOW_OUT;
    }

    return out;
}
```

B.1.4. Estimulador aleatorio (Random Stimulator)

```
double aleatModel (double voltage, double time){
    static double lastTime, spkTime;
    static short numCalls = NEW_TIME_WINDOW;
        static unsigned int jitter, jitter2, lanzado = 0, spkDetected = 0;
        unsigned int i;

    if (stimulator->trigger_function){
        spkDetected = detect_spike(voltage, time);
        if (spkDetected==1) spkTime = time;

        if (numCalls==NEW_TIME_WINDOW){ //New period
                    lanzado = 0;
            spkDetected = 0;
                    get_random_bytes(&i, sizeof(unsigned int));
                    jitter = i % (unsigned int)(wordsShm->randomPeriod*1000 -
    wordsShm->windowTime);
            lastTime = time;
            numCalls++;
            get_random_bytes(&i, sizeof(unsigned int));
```

```
            jitter2 = 10;//(wordsShm->wordSize)*(wordsShm->windowTime) + (i %
    (unsigned int)(wordsShm->windowTime));
        }

        if ((lanzado==0) && (jitter < (unsigned int)(time-lastTime))){
    //jitter passed
            if ((lanzado == 0) && (jitter2 < (unsigned int)(time - spkTime)))
                //&& (((wordsShm->wordSize)*(wordsShm->windowTime) +
    wordsShm->windowTime) > (time - spkTime))){
                        #ifdef __DEBUG__
                        rt_printk ("Info stimulator trigger_function %u <
    %u\n", jitter2, (unsigned int)(time - spkTime));
                        #endif
                        stimulator->trigger_function();
                lanzado=1;
            }
            }

        if ((wordsShm->randomPeriod*1000) < time-lastTime){ //randomPeriod
    passed
                numCalls = NEW_TIME_WINDOW;
        }
    }

        return 0;
}
```

B.2. Módulo Words en espacio de usuario

La implementación del módulo words en espacio de usuario se reduce casi exclusivamente a la creación de los diálogos de la interfaz de usuario y a la comunicación con la tarea en tiempo real mediante la carga de los parámetros introducidos por el usuario en la memoria compartida.

No se incluye el código asociado a la construcción gráfica de la interfaz de usuario, sólo el referente a su implementación.

B.2.1. Memoria Compartida: ShmWords

```
//ShmWords.h
```

```
#ifndef __SHM_WORDS_H__
#define __SHM_WORDS_H__

#include <iostream>
#include "../include/words_shm.h"
#include <fcntl.h>

#include <vector>
#include <qstring.h>

using namespace std;

#define MAX_WORDS 1024

class ShmWords
{

private:
    static ShmWords* pinstance;
    WORDS_SHM *words_shm;

protected:
    ShmWords();

public:
    ~ShmWords();
    bool init();

    // Instantiate the class
    static ShmWords *Instance();

    void activateWordsDetector();
    void deactivateWordsDetector();
    void activateWordsHistogram();
    void deactivateWordsHistogram();
    void activateWordsInformation();
    void deactivateWordsInformation();
    int getActivatedWordsDetector();
    int getActivatedWordsHistogram();
    int getActivatedWordsInformation();

    void setSpikeThreshold(double threshold);
    double getSpikeThreshold();
    void setBitTime(double time);
```

```
    double getBitTime();
    void setWordSize(int wSize);
    double getWordSize();
    void setWord(QString word);
    char* getWord();
    QString* getWord(QString* word);
    void setWindowTime(double windowTime);
    double getWindowTime();
    void setNumWords(int numWords);
    int getNumWords();
    void setSem(int val);
    int getSem();

    void setRandomPeriod(double randomPeriod);
    double getRandomPeriod();

    vector<int>* getHist();
};

#endif
```

```
#include "ShmWords.h"

#ifndef __NO_RT__

#include <rtai_shm.h>
#include <rtai_nam2num.h>

#endif


#include <math.h>
//#include "../../include/words_shm.h"

ShmWords::ShmWords( )
{
} // End ShmWords

ShmWords::~ShmWords ()
{
#ifndef __NO_RT__
    if ( words_shm != 0)
    {
```

```
        rtai_free (nam2num(WORDSSHM), words_shm);// nam2num: Convert a 6
    characters string to an unsigned long,
                                                // to be used as an
    identifier for RTAI services symmetrically
                                                // available in user and
    kernel space
    }
#endif
    pinstance = 0;
} // End ~ShmWords

ShmWords* ShmWords::pinstance = 0; // initialize pointer

ShmWords *ShmWords::Instance()
{
    if (pinstance == 0)
    {
        pinstance = new ShmWords();

        if ( pinstance->init() == false)
        {
            pinstance = 0;
        }
    }

    return pinstance;
} // End instance

bool ShmWords::init ()
{
    cout << "sizeof Topology" << sizeof (words_shm) << endl;
#ifndef __NO_RT__
    //!< A pointer to the shared memory
    if ( !(words_shm = (WORDS_SHM*) rtai_malloc(nam2num(WORDSSHM),
    sizeof(WORDS_SHM))) )
    {
        cout << "ERROR: Failed to connect with shared memory" << endl;
        return false;
    }
#endif

    return true;
}// End init
```

```
void ShmWords::activateWordsDetector(){
    words_shm->activated[0] = 1;
}

void ShmWords::deactivateWordsDetector(){
    words_shm->activated[0] = 0;
}

int ShmWords::getActivatedWordsDetector(){
    return words_shm->activated[0];
}

void ShmWords::activateWordsHistogram(){
    words_shm->activated[1] = 1;
}

void ShmWords::deactivateWordsHistogram(){
    words_shm->activated[1] = 0;
}

int ShmWords::getActivatedWordsHistogram(){
    return words_shm->activated[1];
}

void ShmWords::activateWordsInformation(){
    words_shm->activated[2] = 1;
}

void ShmWords::deactivateWordsInformation(){
    words_shm->activated[2] = 0;
}

int ShmWords::getActivatedWordsInformation(){
    return words_shm->activated[2];
}

void ShmWords::setSpikeThreshold(double threshold)
{
    words_shm->threshold = threshold;
}

double ShmWords::getSpikeThreshold()
{
    return words_shm->threshold;
```

```
}

void ShmWords::setBitTime(double time)
{
    words_shm->bitTime = time;
}

double ShmWords::getBitTime()
{
    return  words_shm->bitTime;
}

void ShmWords::setWordSize(int wordSize){
    words_shm->wordSize = wordSize;
}

double ShmWords::getWordSize(){
    return  words_shm->wordSize;
}

void ShmWords::setWord(QString word){
    unsigned int i;
    for (i=0; i<word.length(); i++){
            words_shm->word[i] = (QChar(word[i])).digitValue();
    }
}

char* ShmWords::getWord(){
    return  words_shm->word;
}

QString* ShmWords::getWord(QString *word){
    int i;
    for (i=0; i<words_shm->wordSize; i++){
        word->append('0'+(words_shm->word[i]));
    }
    return word;
}

void ShmWords::setWindowTime(double windowTime){
    words_shm->windowTime = windowTime;
}

double ShmWords::getWindowTime(){
```

```
    return words_shm->windowTime;
}

void ShmWords::setRandomPeriod(double randomPeriod){
    words_shm->randomPeriod = randomPeriod;
}

double ShmWords::getRandomPeriod(){
    return words_shm->randomPeriod;
}

void ShmWords::setNumWords(int numWords){
    words_shm->numWords = numWords;
}

int ShmWords::getNumWords(){
    return words_shm->numWords;
}

void ShmWords::setSem(int val){
    words_shm->histSem = val;
}

int ShmWords::getSem(){
    return words_shm->histSem;
}

vector<int>* ShmWords::getHist(){
    return new vector<int>(words_shm->hist, words_shm->hist+MAX_WORDS);
}
```

B.2.2. Generador de histograma de palabras (Words Histogram Generator)

```
WordsHistogramImpl::WordsHistogramImpl( QWidget* parent, const char* name,
    bool modal, WFlags fl )
        : WordsHistogram( parent, name, modal, fl ){

    cout << __PRETTY_FUNCTION__ << endl;
    m_shmWords = ShmWords::Instance();

    setupDialog();
```

```
    //Signals
    connect(acceptButton, SIGNAL( clicked() ), this, SLOT( doAccept ()) );
    connect(cancelButton, SIGNAL( clicked() ), this, SLOT( reject ()) );
}

WordsHistogramImpl::~WordsHistogramImpl(){
    cout << __PRETTY_FUNCTION__ << endl;
}

void WordsHistogramImpl::doAccept(){
    cout << __PRETTY_FUNCTION__ << endl;
    Q_ASSERT (m_shmWords != 0);

    int numWords;

        if (10 > QString(bitTimeLineEdit->text()).toDouble()){
                showError("Minimum Bit Time: 10");
                return;
        }
        if (600 < QString(windowTimeLineEdit->text()).toDouble()){
                showError("Maximum Window Time: 600");
                return;
        }
        if (QString(bitTimeLineEdit->text()).toDouble() >
    (QString(windowTimeLineEdit->text()).toDouble()*1000)){
                showError("Window Time less than Bit Time");
                return;
        }

    m_shmWords->activateWordsHistogram();

    m_shmWords->setSpikeThreshold (
    QString(spikeThresholdLineEdit->text()).toDouble() );
    m_shmWords->setBitTime ( QString(bitTimeLineEdit->text()).toDouble() );
    m_shmWords->setWordSize ( wordLengthSpinBox->value() );
    m_shmWords->setWindowTime ( QString(windowTimeLineEdit->text()).toDouble()
    );
        m_shmWords->setSem(0);

    #ifdef __DEBUG__
    cout << "Spike Threshold: " << m_shmWords->getSpikeThreshold() << endl;
    cout << "Window Time (ms): " << m_shmWords->getBitTime() << endl;
    cout << "Word Size: " << m_shmWords->getWordSize() << endl;
```

```
    cout << "Storing Time: " << m_shmWords->getWindowTime() << endl;
    #endif

    numWords = calculateNumWords(m_shmWords->getWindowTime(),
    m_shmWords->getBitTime(), m_shmWords->getWordSize());
    m_shmWords->setNumWords(numWords);
    accept();
}

int WordsHistogramImpl::calculateNumWords(double windowTime, double bitTime,
    int wordSize){
        cout << __PRETTY_FUNCTION__ << endl;
        //cout << (windowTime*1000)/(bitTime)-(wordSize-1) << endl;
    return (windowTime*1000)/(bitTime)-(wordSize-1);
}
```

B.2.3. Detector de palabra binaria o código para estimulación en ciclo cerrado (Word Detector)

```
WordsDetectorImpl::WordsDetectorImpl( QWidget* parent, const char* name, bool
    modal, WFlags fl )
        : WordsDetector( parent, name, modal, fl ){

    cout << __PRETTY_FUNCTION__ << endl;
    m_shmWords = ShmWords::Instance();

    setupDialog();

    //Signals
    connect(acceptButton, SIGNAL( clicked() ), this, SLOT( doAccept ()) );
    connect(cancelButton, SIGNAL( clicked() ), this, SLOT( reject ()) );
}

WordsDetectorImpl::~WordsDetectorImpl(){
    cout << __PRETTY_FUNCTION__ << endl;
}

void WordsDetectorImpl::doAccept(){
    cout << __PRETTY_FUNCTION__ << endl;
    Q_ASSERT (m_shmWords != 0);
```

```
        if (10 > QString(bitTimeLineEdit->text()).toDouble()){
                showError("Minimum Bit Time: 10");
                return;
        }
        if (wordSizeSpinBox->value()!=wordLineEdit->text().length()){
                showError("Incorrect Word Length");
                return;
        }

    m_shmWords->activateWordsDetector();

    m_shmWords->setSpikeThreshold (
    QString(spikeThresholdLineEdit->text()).toDouble() );
    m_shmWords->setBitTime ( QString(bitTimeLineEdit->text()).toDouble() );
    m_shmWords->setWordSize ( wordSizeSpinBox->value() );
    m_shmWords->setWord ( wordLineEdit->text() );

    #ifdef __DEBUG__
    cout << "Spike Threshold: " << m_shmWords->getSpikeThreshold() << endl;
    cout << "Window Time (ms): " << m_shmWords->getBitTime() << endl;
    cout << "Word Size: " << m_shmWords->getWordSize() << endl;
    cout << "Word: " << m_shmWords->getWord(new QString())->local8Bit() <<
    endl;
    #endif

    if (m_shmWords->getNumWords() < 1) m_shmWords->setNumWords(1);
    accept();
}

void WordsDetectorImpl::showError(const char* tag){
                error = new QDialog(this, "errorDialog");

                errorLabel = new QLabel( error, "errorLabel" );
                errorLabel->setGeometry( QRect( 10, 10, 180, 20 ) );
                errorLabel->setFrameShape( QLabel::NoFrame );
                errorLabel->setFrameShadow( QLabel::Plain );

                errorButton = new QPushButton( error, "errorButton" );
                errorButton->setGeometry( QRect( 50, 40, 95, 27 ) );

                error->resize( QSize(199, 76).expandedTo(minimumSizeHint()) );

                errorLabel->setText( tr( tag ) );
                errorButton->setText( tr( "Accept" ) );
```

```
                error->setCaption( tr( "Error" ) );

                connect( errorButton, SIGNAL( clicked() ), error, SLOT(
    close() ) );

                error->show();
                return;
}
```

B.2.4. Estimulador aleatorio (Random Stimulator)

```
//WordsRandom.cpp
WordsRandomImpl::WordsRandomImpl( QWidget* parent, const char* name, bool
    modal, WFlags fl )
        : WordsRandom( parent, name, modal, fl ){

    cout << __PRETTY_FUNCTION__ << endl;
    m_shmWords = ShmWords::Instance();

    setupDialog();

    //Signals
    connect(acceptButton, SIGNAL( clicked() ), this, SLOT( doAccept ()) );
    connect(cancelButton, SIGNAL( clicked() ), this, SLOT( reject ()) );
}

WordsRandomImpl::~WordsRandomImpl(){
    cout << __PRETTY_FUNCTION__ << endl;
}

void WordsRandomImpl::doAccept(){
    cout << __PRETTY_FUNCTION__ << endl;
    Q_ASSERT (m_shmWords != 0);

    int numWords;

    m_shmWords->activateWordsRandom();

    m_shmWords->setSpikeThreshold (
    QString(spikeThresholdLineEdit->text()).toDouble() );
    m_shmWords->setWindowTime ( QString(windowTimeLineEdit->text()).toDouble()
    );
```

```
    m_shmWords->setRandomPeriod (
    QString(randomPeriodLineEdit->text()).toDouble() );
    //m_shmWords->setWordSize ( numZerosSpinBox->value() );

    #ifdef __DEBUG__
    cout << "Spike Threshold: " << m_shmWords->getSpikeThreshold() << endl;
    cout << "Window Time: " << m_shmWords->getWindowTime() << endl;
    cout << "Random Period: " << m_shmWords->getRandomPeriod() << endl;
    cout << "End Zeros: " << m_shmWords->getWordSize() << endl;
    #endif

    numWords=2;
    m_shmWords->setNumWords(numWords);
    accept();
}

void WordsRandomImpl::showError(const char* tag){
                error = new QDialog(this, "errorDialog");

                errorLabel = new QLabel( error, "errorLabel" );
                errorLabel->setGeometry( QRect( 10, 10, 180, 20 ) );
                errorLabel->setFrameShape( QLabel::NoFrame );
                errorLabel->setFrameShadow( QLabel::Plain );

                errorButton = new QPushButton( error, "errorButton" );
                errorButton->setGeometry( QRect( 50, 40, 95, 27 ) );

                error->resize( QSize(199, 76).expandedTo(minimumSizeHint()) );

                errorLabel->setText( tr( tag ) );
                errorButton->setText( tr( "Accept" ) );
                error->setCaption( tr( "Error" ) );

                connect( errorButton, SIGNAL( clicked() ), error, SLOT(
    close() ) );

                error->show();
                return;
}

int WordsRandomImpl::calculateNumWords(double windowTime, double bitTime, int
    wordSize){
        cout << __PRETTY_FUNCTION__ << endl;
        cout << (windowTime*1000)/(bitTime*wordSize) << endl;
```

```
    return (windowTime*1000)/(bitTime*wordSize);
}
```

B.3. Librería de binarización, almacenamiento y gestión de códigos: WordsLib

```
//WordsBuffer.h
#ifndef __WORDS_BUFFER__
#define __WORDS_BUFFER__

#define MAX_WORDS_BUFF 60000

#define NOT_INITIATED -2
#define ERR -1
#define OK 0

#define MAX_BITS_WORD 10
#define MAX_WORDS 1024

typedef int WbElement;

typedef struct
{
    char bits[MAX_BITS_WORD];
    int wordLength;
    int numBits;
    char* init;
    char* insert;
} BitsBuffer;

typedef struct
{
    WbElement words[MAX_WORDS_BUFF];
    BitsBuffer bb;
    int maxWords;
    int numWords;
    WbElement* init;
    WbElement* insert;
    WbElement* check;
} WordsBuffer;
```

```
void wbCreateHistogram (WordsBuffer* wb, int* results, int numWords);
int wbInit(WordsBuffer* wb, int length, int maxWords);
int wbBitInsert(WordsBuffer* wb, char bit);  //Inserts bit on BitBuffer
int wbWordInsert (WordsBuffer* wb, int word);
int wbStoreWord (WordsBuffer *wb); //Transformation Bits 2 Double done here
int wbCheckWordMatch(WordsBuffer *wb, char *word);
int wbBits2Int(WordsBuffer *wb);
int Bits2Int(char* bb, int length);

void bbAdvancePtr (BitsBuffer* bb, char** ptr);
```

```
//WordsBuffer.c
#include <stdio.h>
#include "wordsBuffer.h"

int wbInit(WordsBuffer* wb, int length, int maxWords){
    wb->insert=wb->words;
    wb->init=wb->words;
    wb->check = wb->words;
    wb->bb.wordLength=length;
    wb->numWords=0;
    wb->maxWords=maxWords; //Calculated from the GUI
    wb->bb.init = wb->bb.bits;
    wb->bb.insert = wb->bb.bits;

    return OK;
}

void wbCreateHistogram (WordsBuffer* wb, int* results, int numWords){
        int i;

        for (i=0; i<MAX_WORDS; i++){
                results[i]=0;
        }

        for (i=0; i<numWords; i++){
                results[wb->words[i]]++;
        }
}

int wbBitInsert(WordsBuffer* wb, char bit){  //Inserts bit on BitBuffer
    *(wb->bb.insert)=bit;
    bbAdvancePtr(&(wb->bb), &(wb->bb.insert));
```

```
    wb->bb.numBits++;
    return OK;
}

void bbAdvancePtr (BitsBuffer* bb, char** ptr){
    if (*ptr==&(bb->bits[bb->wordLength-1])) *ptr = bb->bits;
    else (*ptr)++;
}

int wbWordInsert (WordsBuffer* wb, int word){ //Inserts word on WordsBuffer
                                               //Called by StoreWord

    *wb->insert = word; //copy word value
        wb->check=wb->insert;
    wb->numWords++;

    if (wb->insert == wb->words+wb->maxWords){
        wb->insert = wb->words;
        if (wb->init == wb->words+wb->maxWords) wb->init = wb->words;
        else wb->init++;
    } else wb->insert++;

    return OK;
}

int wbStoreWord (WordsBuffer *wb){
    int wordResult;

    if (wb->bb.numBits<wb->bb.wordLength) return ERR;

    wordResult = wbBits2Int(wb);

    //Advance bit init
    bbAdvancePtr(&(wb->bb),&(wb->bb.init));

    //Insert word
    wbWordInsert(wb, wordResult);

    return wordResult;
}

int wbBits2Int(WordsBuffer *wb){
        char *auxPtr;
        int i, exp;
```

```
        int wordResult = 0;

        exp=wb->bb.wordLength - 1;

        auxPtr=wb->bb.init;
    for (i=0; i<wb->bb.wordLength; ++i){
        if ((int)*auxPtr){
            switch (exp){
                case 0: wordResult += 1; break;
                case 1: wordResult += 2; break;
                default: wordResult += pow(2,exp); break;
            }
        }
        exp--;
        bbAdvancePtr(&(wb->bb), &auxPtr);
    }

    return wordResult;
}

int Bits2Int(char* bb, int length){
        char *auxPtr;
        int i, exp;
        int wordResult = 0;

        exp=length - 1;

        auxPtr=bb;
    for (i=0; i<length; ++i){
        if ((int)*auxPtr){
            switch (exp){
                case 0: wordResult += 1; break;
                case 1: wordResult += 2; break;
                default: wordResult += pow(2,exp); break;
            }
        }
        exp--;
        auxPtr++;
    }

    return wordResult;
}

int wbCheckWordMatch(WordsBuffer *wb, char *word){
```

```
        return (*(wb->check)==Bits2Int(word, wb->bb.wordLength));
}
```

Apéndice C

Utilidades

C.1. Utilidad SpikeTimes

Script para Octave encargado de detectar pulsos en un fichero de señal y de devolver los tiempos en que dichos pulsos tienen lugar. Recibe como parámetro de entrada un fichero de adquisición de señal generado con ADClamp y genera un fichero de salida con los tiempos en que se ha detectado pulso.

```
// SpkTimes.cpp
// Extrae los tiempos de spike desde la senial original

#define _FILE_OFFSET_BITS 64

#include <cmath>

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <iomanip>

using namespace std;

float time1;
float Vm\_NEURON1[2];
float NEURON1\_thres,spike;
```

```
unsigned long int i,j,n_col;

void split(const std::string& str, const std::string& delim,
    std::vector<std::string>& parts);

int main(int argc, char* argv[]){
    ifstream inputFile;
    ofstream fp1;
    double initTime;
    string auxStr;

    NEURON1\_thres = 6;

    inputFile.open (argv[1], ios::in | ios::binary);
    inputFile.clear();
        inputFile.seekg(0, ios::beg);

    getline (inputFile, auxStr);
        vector<string> partsAux;
        string space(" ");
        split (auxStr, space, partsAux);
    cout << partsAux[0] << endl;
    initTime = atof(partsAux[0].c_str());

    auxStr = argv[1];
    auxStr += ".spkTimes";
    fp1.open(auxStr.c_str(), ios::out | ios::binary)

    spike=0;
    Vm_NEURON1[1]=0;

    getline (inputFile, auxStr);
    while (!inputFile.eof()){
                vector<string> parts;
        split (auxStr, space, parts);

        Vm_NEURON1[0] = Vm_NEURON1[1];
        Vm_NEURON1[1] = atof(parts[3].c_str());

        time1 = atof(parts[0].c_str())-initTime;

        if(Vm_NEURON1[0]<NEURON1_thres && Vm_NEURON1[1]>NEURON1_thres &&
    (time1-spike > 1)) {
```

```
            spike=time1;
            fp1 << setprecision(9) << time1 << endl;
        }

                getline (inputFile, auxStr);
    }

    inputFile.close();
    fp1.close();

    return(0);
}

void split(const string& str, const string& delim, vector<string>& parts) {
    size_t start, end = 0;
    while (end < str.size()) {
        start = end;
        while (start < str.size() && (delim.find(str[start]) != string::npos))
    {
          start++;  // skip initial whitespace
        }
        end = start;
        while (end < str.size() && (delim.find(str[end]) == string::npos)) {
          end++; // skip to end of word
        }
        if (end-start != 0) {  // just ignore zero-length strings.
          parts.push_back(string(str, start, end-start));
        }
    }
}
```

C.2. Utilidad ImpulseTimes

Script para Octave encargado de detectar impulsos de estimulación en un fichero de señal y de devolver los tiempos en que dichos impulsos son emitidos. Recibe como parámetro de entrada un fichero de adquisición de señal generado con ADClamp y genera un fichero de salida con los tiempos en que se ha detectado pulso.

```
// ImpulseTimes.cpp
// Extrae los tiempos de impulsos desde la senial original

#define _FILE_OFFSET_BITS 64 // le arquivos de tamanho 2^64

#include <cmath>

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <iomanip>

using namespace std;

float time1, prec=0.0001;
float Vm_NEURON1,NEURON1_thres;
int numZeros;

unsigned long int i,j,n_col;

void split(const std::string& str, const std::string& delim,
    std::vector<std::string>& parts);

int main(int argc, char* argv[]){
    ifstream inputFile;
    ofstream fp1;
    float initTime;
    string auxStr;

    NEURON1_thres = 0.1;

    inputFile.open (argv[1], ios::in | ios::binary);
    inputFile.clear();
```

```
        inputFile.seekg(0, ios::beg);

    getline (inputFile, auxStr);
        vector<string> partsAux;
        string space(" ");
        split (auxStr, space, partsAux);

    initTime = atof(partsAux[0].c_str());

    auxStr = argv[1];
    auxStr += ".impulseTimes";
    fp1.open(auxStr.c_str(), ios::out | ios::binary);

    numZeros=0;
    Vm_NEURON1=0;

    while (!inputFile.eof()){
                vector<string> parts;
        split (auxStr, space, parts);

        Vm_NEURON1 = atof(parts[1].c_str());

        if ((Vm_NEURON1>(-prec)) && (Vm_NEURON1<prec)) {
            numZeros++;
        }
        else{
            if ((numZeros>=10) && ((Vm_NEURON1>NEURON1_thres) ||
    (Vm_NEURON1<(-NEURON1_thres)) )) {
                time1 = atof(parts[0].c_str())-initTime;
                fp1 << setprecision(9) << time1 << endl;
                numZeros = 0;
            }
        }

                getline (inputFile, auxStr);
    }

    inputFile.close();
    fp1.close();

    return(0);
}

void split(const string& str, const string& delim, vector<string>& parts) {
```

```
    size_t start, end = 0;
    while (end < str.size()) {
        start = end;
        while (start < str.size() && (delim.find(str[start]) != string::npos))
    {
          start++;  // skip initial whitespace
        }
        end = start;
        while (end < str.size() && (delim.find(str[end]) == string::npos)) {
          end++; // skip to end of word
        }
        if (end-start != 0) {  // just ignore zero-length strings.
          parts.push_back(string(str, start, end-start));
        }
    }
}
```

C.3. Utilidad PreImpulseSpikeDistribution

```
arg_list = argv ();
filename1 = arg_list{1};
impulseTimes = load("-ascii",filename1);
filename2 = arg_list{2};
spikeTimes = load("-ascii",filename2);

[N1,M1] = size(impulseTimes);
[N2,M2] = size(spikeTimes);


preImpulseSpikeTimes = [];

lastj=0;
j=1; k=1;
for i=1:N1
    while (j<=length(spikeTimes) && spikeTimes(j)<=impulseTimes(i))
        j=j+1;
    endwhile

    if (j>1) k=j-1; endif

    if (j<=length(spikeTimes) && j>lastj)
        if (k<=N2)
```

```
            preImpulseSpikeTimes = [preImpulseSpikeTimes; impulseTimes(i),
    spikeTimes(k), impulseTimes(i)-spikeTimes(k)];
            lastj = j;
        endif
    endif
endfor


outputFile = [filename2 ".eps"];

figure(1, "visible", "off");

[nn,xx] = hist(preImpulseSpikeTimes(:,3),0:10:200,100);
nn(11:length(nn)) = 0;
bar(xx,nn);
print (outputFile, "-depsc2");
```

Script para Octave encargado de calcular la distribución de retrasos pulso-estímulo. Recibe un fichero de tiempos de pulso y un fichero de tiempos de impulsos de estimulación y, con ellos, calcula un histograma de retrasos. A partir de dicho cálculo genera y devuelve un gráfico de barras que muestra la distribución de los retrasos.

Apéndice D

Aplicación PezHist: Generación de histogramas y cálculo de entropías offline.

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <cmath>
#include <iomanip>
#include <memory>
#include <fstream>

#include "pezHist.h"
#include "SignalProcessor.h"
#include "ErrorFilter.h"
#include "WordHistGenerator.h"

using namespace std;

//@todo: get config file from command line


//@todo move program logic to specific classes: Keep main only to initialize
    GUI & classes
```

```
int main(int argc, char* argv[]){
    //float entropy = 0.0;
        vector< shared_ptr<ErrorFilter> > EFilters;
        vector< shared_ptr<WordHistGenerator> > HistGens;
    vector<float> binTimes;
        shared_ptr<ProblemConfig> info;

        if ((argc!=2)&&(argc!=4)){
                cout << "Error: Number of arguments does not match
    definition." << endl;
                return 0;
        }

        //@todo: move to ProblemConfig class + exception
        ifstream test(argv[1]);
        if (!test.good()){
                cout << "Error: Bad config file" << endl;
                return 0;
        }

    //@todo: read config filename from command line
    if (argc!=4){
                info = make_shared<ProblemConfig>
    (ProblemConfig("config.yaml"));
        }
        else{
                info = make_shared<ProblemConfig>
    (ProblemConfig("config.yaml", argv[2], argv[3]));
        }

    HDF5HistWriter H5FileWriter(info->getOutputFileName(),
    info->getWordLengths(), info->getBinTimes());

    H5FileWriter.writeProblemInfo(info);


    shared_ptr<SignalProcessor> sp = shared_ptr<SignalProcessor>(new
    SignalProcessor(info));

    for (double bT : info->getBinTimes()){
        shared_ptr<ErrorFilter> errorFilter(new ErrorFilter(bT,
    info->getTotalTime(), sp));
        EFilters.push_back(errorFilter);
        for (int wL : info->getWordLengths()){
```

```
            shared_ptr<WordHistGenerator> wordHistGen(new
    WordHistGenerator(wL,errorFilter));
            HistGens.push_back(wordHistGen);
        }
    }

    sp->run();

    //@todo: move to a class -> printResults
    for (auto histGen : HistGens){
        auto hist = histGen->getHist();
        hsize_t histDims[2];
        histDims[0]=histGen->getMaxWords(); histDims[1]=2;

        pair<float,int> par(histGen->getBinTime(),histGen->getWordLength());

        H5FileWriter.writeHistData(hist, par, histDims);

    H5FileWriter.writeEntropy(histGen->getEntropy()/histGen->getWordLength(),
                                        histGen->getBinTime(),
    histGen->getWordLength());
    }

    for (auto eFilter : EFilters){
        H5FileWriter.writeErrors(eFilter->getErrors(),
    eFilter->getBitErrors(), eFilter->getBinTime());
    }

    return 0;
}
```

```
#ifndef __PEZ_HIST__
#define __PEZ_HIST__

#define BIT_DETECTED_OUT 2
#define BIT_NOT_DETECTED_OUT 0
#define END_WINDOW_OUT 1
#define WORD_DETECTED_OUT 3

#define END_HISTOGRAM_TIME -2

#define FIRST_CALL 0
#define NEW_TIME_WINDOW 1
```

```
#define FALSE 0
#define TRUE 1

#include "HDF5HistWriter.h"
#include "ProblemInfo.h"
#include "ProblemConfig.h"

extern "C" {
        #include "wordsBuffer.h"
}

//template <typename T,unsigned S>
//inline unsigned arraysize(const T (&v)[S]) { return S; }

void readInfoArguments(char* argv[]);
void updateParameters(float binTime, int wordLength);
void print2File(HDF5HistWriter &H5FileWriter, std::vector< std::pair<int,int>
    > pairVector, float entropy);
void fileHistSimulation(WordsBuffer *wb);
void orderHistVector(std::vector<int> &wordsVec, std::vector<
    std::pair<int,int> >& wordsPosVector);
void calculateEntropy(std::vector<int> &probs, int n, float& entropy);
void split(const std::string& str, const std::string& delim,
    std::vector<std::string>& parts);
double histogramModel(WordsBuffer *wb, double currentV, double time, double
    initTime);
int detect_spike (double voltage, double time);
int detect_spike_2 (double voltage, double time);

#endif
```

```
//      SignalProcessor.cpp
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#include <iostream>
#include <memory>
#include "SignalProcessor.h"
#include "UpcrossSpikeDetector.h"
#include "DowncrossSpikeDetector.h"
#include "FormSpikeDetector.h"
```

```
using namespace std;

SignalProcessor::SignalProcessor(shared_ptr<ProblemConfig> info){
    _info = info;

    string type = _info->getSpkDetType();

    if (type.compare("Upcross"))//@todo: constant string
        _spkDetector = unique_ptr<SpikeDetector>(new
    UpcrossSpikeDetector(info->getVolThreshold(),1.0));
    if (type.compare("Downcross"))//@todo: constant string
        _spkDetector = unique_ptr<SpikeDetector>(new
    DowncrossSpikeDetector(info->getVolThreshold(),1.0));
    if (type.compare("FormSpike"))//@todo: constant string
        _spkDetector = unique_ptr<SpikeDetector>(new
    FormSpikeDetector(info->getVolThreshold(),1.0));

}

SignalProcessor::~SignalProcessor(){
}

void SignalProcessor::run(){
    string auxStr;


    //@todo: Check file existence
    _inputFile.open (_info->getInputFileName(), ios::in | ios::binary);
    _inputFile.clear() ;
        _inputFile.seekg(0, ios::beg) ;

    getline (_inputFile, auxStr);
        vector<string> partsAux;
        string space(" ");
        split (auxStr, space, partsAux);

    _initTime = atof(partsAux[0].c_str());

    while (!_inputFile.eof()){
                vector<string> parts;
        split (auxStr, space, parts);
```

```
        if (END_HIST_TIME ==
    histogramModel(atof(parts[_info->getVolColumn()-1].c_str()),
    atof(parts[0].c_str()))){
                        break;
                }

                getline (_inputFile, auxStr);
    }

    _inputFile.close();
}

int SignalProcessor::histogramModel(double currentV, double time){
    int spk = _spkDetector->detectSpike(currentV,time);
    if (spk==1){
        notify(time-_initTime);
    }
    if ((time - _initTime) >= _info->getTotalTime()){
        notify(_info->getTotalTime());
        return END_HIST_TIME;
    }
    return spk;
}

int SignalProcessor::histogramModel(double currentV, double time, double
    endTime){
    int spk = _spkDetector->detectSpike(currentV,time);
    if (spk==1){
        notify(time-_initTime);
    }
    if ((time - _initTime) >= endTime){
        notify(endTime);
        return END_HIST_TIME;
    }
    return spk;
}

void SignalProcessor::notify(double time) {
  for (auto v : _views)
    v->update(time);
}

void SignalProcessor::split(const string& str, const string& delim,
    vector<string>& parts) {
```

```
  size_t start, end = 0;
  while (end < str.size()) {
    start = end;
    while (start < str.size() && (delim.find(str[start]) != string::npos)) {
      start++;  // skip initial whitespace
    }
    end = start;
    while (end < str.size() && (delim.find(str[end]) == string::npos)) {
      end++; // skip to end of word
    }
    if (end-start != 0) {  // just ignore zero-length strings.
      parts.push_back(string(str, start, end-start));
    }
  }
}
```

```
//      SignalProcessor.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __SIGNAL_PROCESSOR_H__
#define __SIGNAL_PROCESSOR_H__

#include <iostream>
#include <vector>
#include <memory>
#include <fstream>

#include "SpikeDetector.h"
#include "SpikesObserver.h"
#include "ProblemConfig.h"

class SignalProcessor
{
private:
    const int END_HIST_TIME = -2;
    std::vector<SpikesObserver*> _views;
        std::ifstream _inputFile;
        double _initTime;
    double _endTime;
    std::unique_ptr<SpikeDetector> _spkDetector;
    std::shared_ptr<ProblemConfig> _info;
```

```
    void split(const std::string& str, const std::string& delim,
    std::vector<std::string>& parts);

public:
    SignalProcessor(std::shared_ptr<ProblemConfig> info);
    virtual ~SignalProcessor();
    void attach(SpikesObserver *obs) {
        _views.push_back(obs);
    }
    void notify(double time);
    void run();
    int histogramModel(double currentV, double time);
    int histogramModel(double currentV, double time, double endTime);
};

#endif /* __SIGNAL_PROCESSOR_H__ */
```

```
//      SpikesObserver.cpp
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>

#include "SpikesObserver.h"
#include "SignalProcessor.h"
#include "BitsObserver.h"

using namespace std;

SpikesObserver::SpikesObserver(shared_ptr<SignalProcessor> mod, double
    binTime) {
    _model = mod;
    _binTime = binTime;
    _model->attach(this);
}

SpikesObserver::~SpikesObserver(){

}

shared_ptr<SignalProcessor> SpikesObserver::getSubject() {
    return _model;
}

void SpikesObserver::notify(int bit) {
```

```
  for (auto v : _views)
    v->update(bit);
}

void SpikesObserver::attach(BitsObserver *obs) {
    _views.push_back(obs);
}
```

```
//      SpikesObserver.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __SPIKE_OBSERVER_H__
#define __SPIKE_OBSERVER_H__

#include <iostream>
#include <vector>
#include <memory>

class SignalProcessor;
class BitsObserver;

class SpikesObserver {
private:
    std::shared_ptr<SignalProcessor> _model;
    std::vector<BitsObserver*> _views;


public:
    SpikesObserver(std::shared_ptr<SignalProcessor> mod, double binTime);
    virtual ~SpikesObserver();
    virtual void update(double time) = 0;
    void attach(BitsObserver *obs);
    double getBinTime(){
        return _binTime;
    }

protected:
    std::shared_ptr<SignalProcessor> getSubject();
    void notify(int bit);
    double _binTime;
};
```

```
#endif /*__SPIKE_OBSERVER_H__*/
```

```
//      BitsObserver.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __BITS_OBSERVER_H__
#define __BITS_OBSERVER_H__

#include <iostream>
#include <memory>
#include <cmath>
#include <SpikesObserver.h>

class BitsObserver
{
private:
    std::shared_ptr<SpikesObserver> _model;
public:
    BitsObserver(std::shared_ptr<SpikesObserver> mod) {
        _model = mod;
        _model->attach(this);
    }
    virtual ~BitsObserver(){}
    virtual void update(int bit) = 0;
protected:
    SpikesObserver* getSubject();
};

#endif /* __BITS_OBSERVER_H__ */
```

```
//      WordHistGenerator.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>

#ifndef __WORD_HIST_GENERATOR_H__
#define __WORD_HIST_GENERATOR_H__

#include <iostream>
#include <cmath>
#include <vector>
```

```
#include <utility>
#include <algorithm>

#include <SpikesObserver.h>
#include <SignalProcessor.h>
#include <BitsObserver.h>
extern "C" {
        #include "wordsBuffer.h"
}

class WordHistGenerator : public BitsObserver
{
private:
    const int BIT_ERROR=-1;
    const int BIT_END=-2;

    int _wordLength;
    int _binTime;
    int _maxWords;
    float _entropy;

    WordsBuffer _wordsbuf;

    std::unique_ptr<int[]> _hist;
    std::vector< std::pair<int,int> > _wordsPosVector;

public:
    WordHistGenerator(int wordLength, std::shared_ptr<SpikesObserver> mod) :
    BitsObserver (mod){
        _binTime = mod->getBinTime();
        _maxWords=(int)pow(2,wordLength);

        _hist = std::unique_ptr<int[]>(new int[_maxWords]());
        for (int i=0; i<_maxWords; i++){
            _hist[i] = 0;
        }
        _wordLength=wordLength;
        wbInit(&_wordsbuf, wordLength, MAX_WORDS_BUFF);
    }

    virtual ~WordHistGenerator(){

    }
```

```
void update(int bit){
    if (bit==BIT_END){
        createHist();
        calculateEntropy();
        return;
    }

    if (bit==BIT_ERROR){
        //wbRestartBitBuff(&_wordsbuf);
        return;
    }

    wbBitInsert(&_wordsbuf,bit);
    wbStoreWord(&_wordsbuf);
}

void createHist(){
    wbCreateHistogram(&_wordsbuf, _hist.get());
    std::vector<int> wordsVec(_hist.get(), _hist.get()+_maxWords);

    for (std::vector<int>::iterator it = wordsVec.begin(); (it !=
wordsVec.end()); ++it){

_wordsPosVector.push_back(std::pair<int,int>(*it,it-wordsVec.begin()));
    }

    std::sort(_wordsPosVector.begin(), _wordsPosVector.end(),
std::greater< std::pair<int,int> > ());
}

void calculateEntropy(){
    _entropy = 0;
    for (std::vector< std::pair<int,int> >::iterator it =
_wordsPosVector.begin(); (it != _wordsPosVector.end()); ++it){
                    float prob = static_cast< float
>(it->first)/_wordsbuf.numWords;
        if ((prob > 0.0000001)||(prob < -0.0000001)){
            _entropy -= prob * (log(prob)/log(2));
            //cout << "----word#: " << *it << " Entropy: " << entropy << "
Prob: " << prob << " log2(p): "<< (log(prob)/log(2)) << endl;
        }
    }
}
```

```
    std::vector< std::pair<int,int> > getHist(){
        return _wordsPosVector;
    }

    int getWordLength(){
        return _wordLength;
    }

    int getBinTime(){
        return _binTime;
    }

    int getMaxWords(){
        return _maxWords;
    }

    float getEntropy(){
        return _entropy;
    }
};

#endif /* __WORD_HIST_GENERATOR_H__ */
```

D.1. Configuración

```
//      problemConfig.cpp
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#include <ProblemConfig.h>

using namespace std;

ProblemConfig::ProblemConfig(string fileName){
    _config = YAML::LoadFile(fileName);
}

ProblemConfig::ProblemConfig(string fileName, string inputFile, string
    outputFile):

                                                    _inputFile(inputFile),
```

```
                                                    _outputFile(outputFile){
    _config = YAML::LoadFile(fileName);
}

ProblemConfig::~ProblemConfig(){

}

double ProblemConfig::getVolThreshold(){
    double vThr;
    vThr = _config["volThreshold"].as<double>();
    return vThr;
}

int ProblemConfig::getVolColumn(){
    int vCol;
    vCol = _config["volColumn"].as<int>();
    return vCol;
}

double ProblemConfig::getTotalTime(){
    double tTime;
    tTime = _config["totalTime"].as<double>() * 1000;
    return tTime;
}

std::string ProblemConfig::getSpkDetType(){
    return _config["spikeDetector"].as<string>();
}

vector<int> ProblemConfig::getWordLengths(){
    vector<int> vAux;
    for(YAML::const_iterator it=_config["wordLengths"].begin();
    it!=_config["wordLengths"].end();++it)
        vAux.push_back(it->as<int>());
    return vAux;
}

vector<float> ProblemConfig::getBinTimes(){
    vector<float> vAux;
    for(YAML::const_iterator it=_config["binTimes"].begin();
    it!=_config["binTimes"].end();++it)
        vAux.push_back(it->as<float>());
```

```
    return vAux;
}

int ProblemConfig::getNumWL(){
    return _config["wordLengths"].size();
}

string  ProblemConfig::getInputFileName(){
        if (_inputFile.empty())
                return _config["inputFile"].as<string>();
        else
                return _inputFile;
}

string  ProblemConfig::getOutputFileName(){
        if (_outputFile.empty())
                return _config["outputFile"].as<string>();
        else
                return _outputFile;
}

int ProblemConfig::getNumBT(){
    return _config["binTimes"].size();
}
```

```
//      problemConfig.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __PROBLEM_CONFIG_H__
#define __PROBLEM_CONFIG_H__

#include <iostream>
#include <fstream>
#include <yaml.h>
#include <string>

class ProblemConfig
{
private:
        YAML::Node _config;
    std::string _inputFile;
```

```
        std::string _outputFile;

public:
        ProblemConfig(std::string fileName);
        ProblemConfig(std::string fileName, std::string inputFile, std::string
    outputFile);

        virtual ~ProblemConfig();

    double getVolThreshold();
    std::vector<int> getWordLengths();
    std::vector<float> getBinTimes();
    std::string getSpkDetType();
    int getNumWL();
    int getNumBT();
    std::string getInputFileName();
    std::string getOutputFileName();
    double getTotalTime();
    int getVolColumn();
};

#endif /* __PROBLEM_CONFIG_H__ */
```

```
#ifndef __PROBLEM_INFO__
#define __PROBLEM_INFO__

typedef struct{
        const char* inputFileName, *outputFileName;
        int wordLength, volColumn, numBT, numWL, maxWords;
        double vThreshold, binTime, maxTime;
} ProblemInfo;

#endif
```

D.1.1. Ejemplo fichero YAML

```
%YAML 1.2
---

#I/O files

#inputFile:
```

```
#outputFile:

#Spike Detector: Upcross, Downcross or FormSpike
spikeDetector : Upcross

#Problem execution data
volColumn: 4
totalTime: 1470
volThreshold: 6
wordLengths: [2,3,4,5]
binTimes:
    [20.0,30.0,40.0,50.0,60.0,70.0,80.0,90.0,100.0,110.0,120.0,130.0,140.0,150.0,
      160.0,170.0,180.0,190.0,200.0,210.0]

...
```

D.2. Detectores de pulsos

```
//      SpikeDetector.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __SPIKE_DETECTOR_H__
#define __SPIKE_DETECTOR_H__

#include <iostream>

class SpikeDetector
{
protected:
    double _filterTime;
    double _vThreshold;
    double _lastSpkTime = 0.0;

public:
    virtual int detectSpike (double voltage, double time) = 0;
};

#endif /* __SPIKE_DETECTOR_H__ */
```

```
//      UpcrossSpikeDetector.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __UPCROSS_SPIKE_DETECTOR_H__
#define __UPCROSS_SPIKE_DETECTOR_H__

#include <iostream>
#include <SpikeDetector.h>

class UpcrossSpikeDetector : public SpikeDetector
{
private:
        double _lastVol1;

public:
        UpcrossSpikeDetector(double vThreshold, double filterTime);
        virtual ~UpcrossSpikeDetector();
    int detectSpike (double voltage, double time);
};

UpcrossSpikeDetector::UpcrossSpikeDetector(double vThreshold= 7.0, double
    filterTime = 1.0){
    _lastVol1=0.0;
    _vThreshold = vThreshold;
    _filterTime = filterTime;
}

UpcrossSpikeDetector::~UpcrossSpikeDetector(){

}

int UpcrossSpikeDetector::detectSpike (double voltage, double time){
    int spike=0;

    if ((_lastVol1>=_vThreshold) &&
        (voltage<_vThreshold)&&
        (time-_lastSpkTime > _filterTime))
    {
        spike=1;
        _lastSpkTime = time;
    }
```

```
    _lastVol1 = voltage;

    return spike;
}
#endif /* __UPCROSS_SPIKE_DETECTOR_H__ */
```

```
//      DowncrossSpikeDetector.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>

#ifndef __DOWNCROSS_SPIKE_DETECTOR_H__
#define __DOWNCROSS_SPIKE_DETECTOR_H__

#include <iostream>
#include <SpikeDetector.h>

class DowncrossSpikeDetector : public SpikeDetector
{
private:
        double _lastVol1=0.0;

public:
    DowncrossSpikeDetector(double vThreshold, double filterTime);
    virtual ~DowncrossSpikeDetector();
    int detectSpike (double voltage, double time);
};

DowncrossSpikeDetector::DowncrossSpikeDetector(double vThreshold= 7.0, double
    filterTime = 1.0){
    _vThreshold = vThreshold;
    _filterTime = filterTime;
}

DowncrossSpikeDetector::~DowncrossSpikeDetector(){

}

int DowncrossSpikeDetector::detectSpike (double voltage, double time){
    int spike=0;

    if ((_lastVol1>=_vThreshold) &&
        (voltage<_vThreshold) &&
        (time-_lastSpkTime > _filterTime))
```

```
    {
        spike=1;
        _lastSpkTime = time;
    }

    _lastVol1 = voltage;

    return spike;
}

#endif /* __DOWNCROSS_SPIKE_DETECTOR_H__ */
```

```
//      FormSpikeDetector.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>

#ifndef __FORM_SPIKE_DETECTOR_H__
#define __FORM_SPIKE_DETECTOR_H__

#include <iostream>
#include <SpikeDetector.h>

class FormSpikeDetector : public SpikeDetector
{
private:
        double _lastVol2=0.0, _lastVol1=0.0;

public:
    FormSpikeDetector(double vThreshold, double filterTime);
    virtual ~FormSpikeDetector();
    int detectSpike (double voltage, double time);
};

FormSpikeDetector::FormSpikeDetector(double vThreshold = 7.0, double
    filterTime = 1.0){
    _vThreshold = vThreshold;
    _filterTime = filterTime;
}

FormSpikeDetector::~FormSpikeDetector(){

}
```

```
int FormSpikeDetector::detectSpike (double voltage, double time){
    int spike=0;

    if ((_lastVol2<_lastVol1) &&
        (_lastVol1>voltage) &&
        (_lastVol1>_vThreshold) &&
        (time-_lastSpkTime > _filterTime))
    {
        spike=1;
        _lastSpkTime=time;
    }

    _lastVol2 = _lastVol1;
    _lastVol1 = voltage;

    return spike;
}

#endif /* __FORM_SPIKE_DETECTOR_H__ */
```

D.3. Tratamiento de Errores

```
//      ErrorFilter.cpp
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#include <iostream>
#include <cmath>
#include <ErrorFilter.h>
#include <SpikesObserver.h>

using namespace std;

ErrorFilter::ErrorFilter(double binTime, double endTime,
    shared_ptr<SignalProcessor> mod) : SpikesObserver(mod, binTime){
    _endTime = endTime;
}

void ErrorFilter::update(double time){
    if (time == _endTime){
        notify (BIT_END);
```

```
        return;
    }

    if (floor(time/_binTime) < _lastBitNum){
        cout << "<ErrorFilter> UPDATE ERROR: Decreasing time." << endl;
    }

    if (floor(time/_binTime) == _lastBitNum){
        _errors++;
        if (_lastBit != BIT_ERROR) _bitErrors++;
        _lastBit = BIT_ERROR;
    }

    if (floor(time/_binTime) > _lastBitNum){
        notify(_lastBit);

        for (int i = 1; i < (floor(time/_binTime) - _lastBitNum); i++){
            notify(0);
        }

        _lastBit = 1;
        _lastBitNum = floor(time/_binTime);
    }
}
```

```
//      ErrorFilter.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>


#ifndef __ERROR_FILTER_H__
#define __ERROR_FILTER_H__

#include <iostream>
#include <memory>
#include <cmath>
#include <SpikesObserver.h>
#include <SignalProcessor.h>

class ErrorFilter : public SpikesObserver
{
private:
    const int BIT_ERROR=-1;
```

```
    const int BIT_END=-2;

    double _endTime;
    long int _lastBitNum = -1;
    int _lastBit = -1;

    int _errors=0;
    int _bitErrors=0;

public:
    ErrorFilter(double binTime, double endTime,
    std::shared_ptr<SignalProcessor> mod);
    virtual ~ErrorFilter(){}

    void update(double time);

    int getErrors(){
        return _errors;
    }

    int getBitErrors(){
        return _bitErrors;
    }
};

#endif /* __ERROR_FILTER_H__ */
```

D.4. Output

```
//      HDF5HistWriter.cpp
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>

#include <iostream>
#include <algorithm>
#include <vector>
#include <utility>
#include <boost/lexical_cast.hpp>

#include "HDF5HistWriter.h"

using namespace std;
```

```
using namespace libStats;

HDF5HistWriter::HDF5HistWriter(std::string fileName, vector<int> wordLengths,
    vector<float> binTimes) :

    _entropyStats(fileName, "Entropy"), _errorsStats(fileName,
    "Errors"),_fileName(fileName){
        hsize_t wordLengthDims[2];
        hsize_t binTimeDims[2];
        hsize_t entropyDims[2];
    hsize_t errorDims[2];

        binTimeDims[0]=binTimes.size();                 binTimeDims[1]=1;
        wordLengthDims[0]=wordLengths.size();           wordLengthDims[1]=1;
        entropyDims[0]=binTimeDims[0]*wordLengthDims[0];
    entropyDims[1]=3;
        errorDims[0]=binTimeDims[0];    errorDims[1]=3;

    //todo: Move to problem info
        _binTimeStreamer = _entropyStats.get_matrix_streamer(binTimeDims,
    "BinTimes");
        _wordLengthStreamer =
    _entropyStats.get_matrix_streamer(wordLengthDims, "WordLengths");

        _entropyStreamer = _entropyStats.get_matrix_streamer(entropyDims,
    "Data");

    _errorsStreamer = _errorsStats.get_matrix_streamer(errorDims, "Data");

        for (int i : wordLengths){
                *_wordLengthStreamer << i;
        }

        for (float i:binTimes){
                *_binTimeStreamer << i;
        }
}

HDF5HistWriter::~HDF5HistWriter(){
        _entropyStats.dump();
    _errorsStats.dump();
}

void HDF5HistWriter::writeProblemInfo(shared_ptr<ProblemConfig> info){
```

```
        hsize_t auxDims[2] = {1, 1};
        libStats::Statistics stats(_fileName, "ProblemInfo");
        auto streamer = stats.get_matrix_streamer(auxDims, "vThreshold");
        *streamer << info->getVolThreshold();
        streamer = stats.get_matrix_streamer(auxDims, "maxTime");
        *streamer << info->getTotalTime();
        streamer = stats.get_matrix_streamer(auxDims, "volColumn");
        *streamer << info->getVolColumn();
        stats.dump();
}

void HDF5HistWriter::writeBintime(float binTime){
        *_binTimeStreamer << binTime;
}

void HDF5HistWriter::writeWordLength(int wordLength){
        *_wordLengthStreamer << wordLength;
}

void HDF5HistWriter::writeEntropy(float entropy, float binTime, int
    wordLength){
        *_entropyStreamer << entropy << binTime << wordLength;
}

void HDF5HistWriter::writeHistData(histVector orderedHist, std::pair<float,
    int> GroupName, hsize_t histDims[2]){
        libStats::Statistics stats(_fileName, string("Hist_" +
    boost::lexical_cast<string>(GroupName.first) + "_" +
    boost::lexical_cast<string>(GroupName.second)));

        auto streamer = stats.get_matrix_streamer(histDims, "HistData");
        for(pair<int,int> words_pair : orderedHist){
                if (words_pair.first==0) break;
                //cout << words_pair.second << " " << words_pair.first << endl;
                *streamer << words_pair.second << words_pair.first;
    //second=Word  first=#
        }
        stats.dump();
}

void HDF5HistWriter::writeErrors(int errors, int bitErrors, float binTime){
    *_errorsStreamer << errors << bitErrors << binTime;
}
```

```
//      HDF5HistWriter.h
//
//      Copyright 2014 Angel Lareo <angel.lareo@gmail.com>

//@todo Doxygen comments
#ifndef __HDF5_HIST_WRITER_H__
#define __HDF5_HIST_WRITER_H__

#include "H5Cpp.h"
#include "Statistics.h"
#include "MatrixStreamer.h"
#include "ProblemConfig.h"

class HDF5HistWriter
{
private:
        libStats::Statistics _entropyStats;
        libStats::Statistics _errorsStats;
        std::string _fileName;
        libStats::MatrixStreamer::SharedPtr _entropyStreamer;
        libStats::MatrixStreamer::SharedPtr _binTimeStreamer;
        libStats::MatrixStreamer::SharedPtr _wordLengthStreamer;
    libStats::MatrixStreamer::SharedPtr _errorsStreamer;

public:
        typedef std::vector< std::pair<int,int> > histVector;
        typedef std::pair<float, int> floatIntPair;

        HDF5HistWriter(std::string fileName, std::vector<int> wordLengths,
    std::vector<float> binTimes);
        virtual ~HDF5HistWriter();

        void writeProblemInfo(std::shared_ptr<ProblemConfig> info);

        void writeBintime(float binTime);
        void writeWordLength(int wordLength);

        void writeEntropy(float entropy, float binTime, int wordLength);
    void writeErrors(int errors, int bitErrors, float binTime);
        void writeHistData(histVector orderedHist, floatIntPair GroupName,
    hsize_t histDims[2]);
};
```

```
#endif //__HDF5_HIST_WRITER_H__
```

Bibliografía

[1] Pablo Chamorro, Carlos Muñiz, Rafael Levi, David Arroyo, Francisco B. Rodríguez, and Pablo Varona. Generalization of the dynamic clamp concept in neurophysiology and behavior. *PLoS ONE*, 7(7):e40887+, July 2012. ISSN 1932-6203. doi: 10.1371/journal.pone.0040887. URL http://dx.doi.org/10.1371/journal.pone.0040887.

[2] Theodore H. Bullock. Electroreception, 2005. URL http://www.worldcat.org/isbn/9780387282756.

[3] Walter Geller. A toxicity warning monitor using the weakly electric fish, gnathonemus petersi. *Water Research*, 18(10):1285–1290, January 1984. ISSN 00431354. doi: 10.1016/0043-1354(84)90034-4. URL http://dx.doi.org/10.1016/0043-1354(84)90034-4.

[4] Peter Cain, William Gerin, and Peter Moller. Short-range navigation of the weakly electric fish, gnathonemus petersii l. (mormyridae, teleostei), in novel and familiar environments. *Ethology*, 96(1):33–45, January 1994. doi: 10.1111/j.1439-0310.1994.tb00879.x. URL http://dx.doi.org/10.1111/j.1439-0310.1994.tb00879.x.

[5] Gerhard von der Emde, Stephan Schwarz, Leonel Gomez, Ruben Budelli, and Kirsty Grant. Electric fish measure distance in the dark. *Nature*, 395(6705):890–894, October 1998. ISSN 0028-0836. doi: 10.1038/27655. URL http://dx.doi.org/10.1038/27655.

[6] von der Emde G. Active electrolocation of objects in weakly electric fish. *The Journal of experimental biology*, 202(# (Pt 10)):1205–1215, May 1999. ISSN 1477-9145. URL http://view.ncbi.nlm.nih.gov/pubmed/10210662.

[7] F. Boyer, P. B. Gossiaux, B. Jawad, V. Lebastard, and M. Porez. Model for a sensor inspired by electric fish. *Robotics, IEEE Transactions on*, 28(2):492–505, April 2012. ISSN 1552-3098. doi: 10.1109/tro.2011.2175764. URL http://dx.doi.org/10.1109/tro.2011.2175764.

[8] A. Awaïss. Gnathonemus petersii, 2010. URL www.iucnredlist.org.

[9] CarolineG Forlim, Carlos Muñiz, ReynaldoD Pinto, FranciscoB Rodríguez, and Pablo Varona. Behavioral driving through on line monitoring and activity-dependent stimulation in weakly electric fish. 14(1):1–2, 2013. doi: 10.1186/1471-2202-14-s1-p405. URL http://dx.doi.org/10.1186/1471-2202-14-s1-p405.

[10] Bruce A. Carlson. Electric signaling behavior and the mechanisms of electric organ discharge production in mormyrid fish. *Journal of physiology, Paris*, 96(5-6): 405–419, 2002. ISSN 0928-4257. URL http://view.ncbi.nlm.nih.gov/pubmed/14692489.

[11] Bruce A. Carlson and Jason R. Gallant. From sequence to spike to spark: evo-devo-neuroethology of electric communication in mormyrid fishes. *Journal of neurogenetics*, 27(3):106–129, September 2013. ISSN 1563-5260. doi: 10.3109/01677063.2013.799670. URL http://dx.doi.org/10.3109/01677063.2013.799670.

[12] Caroline G. Forlim and Reynaldo D. Pinto. Automatic realistic real time stimulation/recording in weakly electric fish: Long time behavior characterization in freely swimming fish and stimuli discrimination. *PLoS ONE*, 9(1):e84885+, January 2014. doi: 10.1371/journal.pone.0084885. URL http://dx.doi.org/10.1371/journal.pone.0084885.

[13] Alain Destexhe. Dynamic-clamp from principles to applications, 2009. URL http://www.worldcat.org/isbn/9780387892795.

[14] J. A. Alves-Gomes. The evolution of electroreception and bioelectrogenesis in teleost fish: a phylogenetic perspective. *Journal of Fish Biology*, 58(6):1489–1511, June 2001. doi: 10.1111/j.1095-8649.2001.tb02307.x. URL http://dx.doi.org/10.1111/j.1095-8649.2001.tb02307.x.

[15] G. K. H. Zupanc. *Electrocommunication*, pages 839–848. Elsevier, 2009. ISBN 978-0-08-045046-9. doi: 10.1016/b978-008045046-9.01956-2. URL http://dx.doi.org/10.1016/b978-008045046-9.01956-2.

[16] H. W. Lissmann. On the function and evolution of electric organs in fish. *Journal of Experimental Biology*, 35(1):156–191, March 1958. ISSN 1477-9145. URL http://jeb.biologists.org/content/35/1/156.abstract.

[17] Angel A. Caputi, María E. Castelló, Pedro Aguilera, and Omar Trujillo-Cenóz. Electrolocation and electrocommunication in pulse gymnotids: signal carriers, pre-receptor mechanisms and the electrosensory mosaic. *Journal of physiology, Paris*, 96(5-6):493–505, 2002. ISSN 0928-4257. URL http://view.ncbi.nlm.nih.gov/pubmed/14692497.

[18] T Szabo and A Fessard. Physiology of electroreceptors. In *Electroreceptors and Other Specialized Receptors in Lower Vertrebrates*, pages 59–124. Springer, 1974.

[19] G. W. Westby. Comparative studies of the aggressive behaviour of two gymnotid electric fish (gymnotus carapo and hypopomus artedi). *Animal behaviour*, 23(1): 192–213, February 1975. ISSN 0003-3472. URL http://view.ncbi.nlm.nih.gov/pubmed/1171643.

[20] A. Caputi and R. Budelli. The electric image in weakly electric fish: I. a data-based model of waveform generation in gymnotus carapo. *Journal of computational neuroscience*, 2(2):131–147, June 1995. ISSN 0929-5313. URL http://view.ncbi.nlm.nih.gov/pubmed/8521283.

[21] Gerhard von der Emde. Discrimination of objects through electrolocation in the weakly electric fish, gnathonemus petersii. 167(3):413–421, 1990. doi: 10.1007/bf00192576. URL http://dx.doi.org/10.1007/bf00192576.

[22] Marielle Thomas, André Florion, Didier Chrétien, and Denis Terver. Real-time biomonitoring of water contamination by cyanide based on analysis of the continuous electric signal emitted by a tropical fish: Apteronotus albifrons. *Water Research*, 30(12):3083–3091, December 1996. ISSN 00431354. doi: 10.1016/s0043-1354(96)00190-x. URL http://dx.doi.org/10.1016/s0043-1354(96)00190-x.

[23] A. A. Caputi, R. Budelli, K. Grant, and C. C. Bell. The electric image in weakly electric fish: physical images of resistive objects in gnathonemus petersii. *Journal of Experimental Biology*, 201(14):2115–2128, July 1998. ISSN 1477-9145. URL http://jeb.biologists.org/content/201/14/2115.abstract.

[24] Gerhard von der Emde and Steffen Fetz. Distance, shape and more: recognition of object features during active electrolocation in a weakly electric fish. *The Journal of experimental biology*, 210(Pt 17):3082–3095, September 2007. ISSN 0022-0949. URL http://view.ncbi.nlm.nih.gov/pubmed/17704083.

[25] Jacob Engelmann, João Bacelo, Michael Metzen, Roland Pusch, Beatrice Bouton, Adriana Migliaro, Angel Caputi, Ruben Budelli, Kirsty Grant, and Gerhard von der Emde. Electric imaging through active electrolocation: implication for the analysis of complex scenes. *Biological Cybernetics*, 98(6):519–539, June 2008. ISSN 0340-1200. doi: 10.1007/s00422-008-0213-5. URL http://dx.doi.org/10.1007/s00422-008-0213-5.

[26] Gerhard von der Emde, Monique Amey, Jacob Engelmann, Steffen Fetz, Caroline Folde, Michael Hollmann, Michael Metzen, and Roland Pusch. Active electrolocation in gnathonemus petersii: behaviour, sensory performance, and receptor systems. *Journal of physiology, Paris*, 102(4-6):279–290, 2008. ISSN 0928-4257. URL http://view.ncbi.nlm.nih.gov/pubmed/18992334.

[27] Gerhard von der Emde, Katharina Behr, Béatrice Bouton, Jacob Engelmann, Steffen Fetz, and Caroline Folde. 3-dimensional scene perception during active electrolocation in a weakly electric pulse fish. *Frontiers in behavioral neuroscience*, 4, 2010. ISSN 1662-5153. doi: 10.3389/fnbeh.2010.00026. URL http://dx.doi.org/10.3389/fnbeh.2010.00026.

[28] G. W. Max Westby. Electrical communication and jamming avoidance betwen resting gymnotus carapo. 4(4):381–393, 1979. doi: 10.1007/bf00303244. URL http://dx.doi.org/10.1007/bf00303244.

[29] Martin Cuddy, Nadia Aubin-Horth, and Rüdiger Krahe. Electrocommunication behaviour and non invasively-measured androgen changes following induced seasonal breeding in the weakly electric fish, apteronotus leptorhynchus. *Hormones and behavior*, 61(1):4–11, January 2012. ISSN 1095-6867. URL http://view.ncbi.nlm.nih.gov/pubmed/21944946.

[30] Bernd Kramer. Electric and motor responses of the weakly electric fish, gnathonemus petersii (mormyridae), to play-back of social signals. 6(1):67–79, 1979. doi: 10.1007/bf00293246. URL http://dx.doi.org/10.1007/bf00293246.

[31] C. D. Hopkins. Neuroethology of electric communication. *Annual Review of Neuroscience*, 11(1):497–535, 1988. doi: 10.1146/annurev.ne.11.030188.002433. URL http://dx.doi.org/10.1146/annurev.ne.11.030188.002433.

[32] B. Kramer. *Electrocommunication in teleost fishes : behavior and experiments*. Springer-Verlag, 1990. ISBN 3540519270. URL http://www.worldcat.org/isbn/3540519270.

[33] P. J. DeCoursey. Sensory perception and communication in electric fish. *Tested Studies for Laboratory Teaching, vol. 5*, 1993.

[34] Bernd Kramer. *Electroreception and communication in fishes : 1 table*. Fischer, 1996. ISBN 3437250388. URL http://www.worldcat.org/isbn/3437250388.

[35] Angel Ariel A. Caputi and Javier Nogueira. Identifying self- and nonself-generated signals: lessons from electrosensory systems. *Advances in experimental medicine and biology*, 739:107–125, 2012. ISSN 0065-2598. URL http://view.ncbi.nlm.nih.gov/pubmed/22399398.

[36] Stephen M. Kajiura and Kim N. Holland. Electroreception in juvenile scalloped hammerhead and sandbar sharks. *Journal of Experimental Biology*, 205(23): 3609–3621, December 2002. ISSN 1477-9145. URL http://jeb.biologists.org/content/205/23/3609.abstract.

[37] Ana Carolina C. Pereira and Angel Ariel A. Caputi. Imaging in electrosensory systems. *Interdisciplinary sciences, computational life sciences*, 2(4):291–307, December 2010. ISSN 1867-1462. URL http://view.ncbi.nlm.nih.gov/pubmed/21153776.

[38] Claudine Teyssedre and Jacques Serrier. Temporal spacing of signals in communication, studied in weakly-electric mormyrid fish (teleostei, pisces). *Behavioural Processes*, 12(1):77–98, January 1986. ISSN 03766357. doi: 10.1016/0376-6357(86)90073-2. URL http://dx.doi.org/10.1016/0376-6357(86)90073-2.

[39] Christa A. Baker, Tsunehiko Kohashi, Ariel M. Lyons-Warren, Xiaofeng Ma, and Bruce A. Carlson. Multiplexed temporal coding of electric communication signals in mormyrid fishes. *The Journal of experimental biology*, 216(Pt 13):2365–2379, July 2013. ISSN 1477-9145. URL http://view.ncbi.nlm.nih.gov/pubmed/23761462.

[40] George Marmont. Studies on the axon membrane. i. a new method. *J. Cell. Comp. Physiol.*, 34(3):351–382, December 1949. doi: 10.1002/jcp.1030340303. URL http://dx.doi.org/10.1002/jcp.1030340303.

[41] Cole K. Electrochemistry in biology and medicine. theodore shedlovsky, editor. john wiley & sons, inc., new york; chapman & hall, ltd., london, 1955. xii + 369 pp. 16.5 x 23.7 cm. price $10.50. *J. Pharm. Sci.*, 44(9):582, September 1955. doi: 10.1002/jps.3030440923. URL http://dx.doi.org/10.1002/jps.3030440923.

[42] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[43] Rudolf Ahlswede and Imre Csiszar. Common randomness in information theory and cryptography. part i: secret sharing. *IEEE Transactions on Information Theory*, 39 (4), 1993.

[44] F. B. Rodriguez, P. Varona, R. Huerta, M. I. Rabinovich, and HenryD Abarbanel. Richer network dynamics of intrinsically non-regular neurons measured through mutual information. In José Mira and Alberto Prieto, editors, *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, volume 2084 of *Lecture Notes in Computer Science*, pages 490–497. Springer Berlin Heidelberg, 2001. doi:

10.1007/3-540-45720-8_58. URL http://dx.doi.org/10.1007/3-540-45720-8_58.

[45] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[46] Giulio Tononi, Olaf Sporns, and Gerald M Edelman. A measure for brain complexity: relating functional segregation and integration in the nervous system. *Proceedings of the National Academy of Sciences*, 91(11):5033–5037, 1994.

[47] Caroline G. Forlim, Reynaldo D. Pinto, Pablo Varona, and Francisco B. Rodriguez. Delay-dependent response in weakly electric fish under closed-loop pulse stimulation. 2014.

[48] Carlos Muñiz, Sara Arganda, Francisco Borja Rodríguez, and GonzaloG Polavieja. Realistic stimulation through advanced dynamic-clamp protocols. In José Mira and JoséR Álvarez, editors, *Mechanisms, Symbols, and Models Underlying Cognition*, volume 3561 of *Lecture Notes in Computer Science*, pages 95–105. Springer Berlin Heidelberg, 2005. doi: 10.1007/11499220_10. URL http://dx.doi.org/10.1007/11499220_10.

[49] Carlos Muniz, Caroline Forlim, Rafael Guariento, Reynaldo Pinto, Francisco Rodriguez, and Pablo Varona. Online video tracking for activity-dependent stimulation in neuroethology. *BMC Neuroscience*, 12(Suppl 1):P358+, 2011. ISSN 1471-2202. doi: 10.1186/1471-2202-12-s1-p358. URL http://dx.doi.org/10.1186/1471-2202-12-s1-p358.

[50] RD Pinto, RC Elson, A Szücs, MI Rabinovich, AI Selverston, and HDI Abarbanel. Extended dynamic clamp: controlling up to four neurons using a single desktop computer and interface. *Journal of neuroscience methods*, 108(1):39–48, 2001.

[51] Pablo Varona, Joaquín J. Torres, Henry D. I. Abarbanel, Mikhail I. Rabinovich, and Robert C. Elson. Dynamics of two electrically coupled chaotic neurons: Experimental observations and model analysis. pages 91–101, 2001.

[52] Jacobo Fernandez-Vargas, Hanns U. Pfaff, Francisco B. Rodríguez, and Pablo Varona. Assisted closed-loop optimization of ssvep-bci efficiency. *Frontiers in neural circuits*, 7, 2013. ISSN 1662-5110. URL http://view.ncbi.nlm.nih.gov/pubmed/23443214.

[53] Carlos Muñiz, Rafael Levi, Meriem Benkrid, Francisco B Rodríguez, and Pablo Varona. Real-time control of stepper motors for mechano-sensory stimulation. *Journal of neuroscience methods*, 172(1):105–111, 2008.

[54] Victor Hugo García García. Desarrollo de una "toolbox" para condicionar el comportamiento de peces eléctricos, basado en la codificación de las señales eléctricas emitidas por éste y su comportamiento.

[55] Thomas Nowotny, Attila Szucs, Reynaldo D. Pinto, and Allen I. Selverston. Stdpc: a modern dynamic clamp. *Journal of neuroscience methods*, 158(2):287–299, December 2006. ISSN 0165-0270. URL http://view.ncbi.nlm.nih.gov/pubmed/16846647.